# ESTIMATION FULFILLMENT IN SOFTWARE DEVELOPMENT PROJECTS

Inaugural-Dissertation
zur Erlangung des Doktorgrades
des Fachbereichs Wirtschaftswissenschaften
der Johann-Wolfgang-Goethe-Universität
Frankfurt am Main

vorgelegt von
Matthias Biggeleben
aus Münster

Frankfurt am Main, 2011

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# List of Abbreviations

| | |
|---|---|
| ACAP | Analyst Capability *(EM; COCOMO)* |
| ACT | Annual Change Traffic |
| AEXP | Applications Experience *(EM; COCOMO)* |
| AFP | Adjusted Function Point |
| CASE | Computer-Aided Software Engineering |
| CBR | Case-Based Reasoning |
| CET | Central European Time |
| CFP | COSMIC Function Point |
| CFSU | COSMIC Functional Size Unit |
| CHF | Swiss *(Confoederatio Helvetica)* Franc |
| CM | Change Management |
| CMM | Capability Maturity Model |
| CMMI | Capability Maturity Model Integration |
| CMS | Competence Measurement System |
| COCOMO | Constructive Cost Model *(Version I; published in 1981)* |
| COCOMO II | Constructive Cost Model *(Version II; published in 2000)* |
| COCOMO'81 | Constructive Cost Model *(Version I; published in 1981)* |
| COSMIC | Common Software Measurement International Consortium |
| COTS | Commercial off-the-Shelf *(Software)* |
| CPLX | Product Complexity *(EM; COCOMO)* |
| CPM | Critical Path Method |
| DATA | Database Size *(EM; COCOMO)* |
| DAX | German Stock Index *("Deutscher Aktienindex")* |
| DI | Degree of Influence |
| DOCU | Documentation Match *(EM; COCOMO)* |
| DRY | *"Don't Repeat Yourself"* (Principle) |
| DSI | Delivered Source Instructions |
| DSL | Domain-Specific Language |
| DSLOC | Delivered Source Lines of Code |
| EAF | Effort Adjustment Factor |
| ECF | Environmental Complexity Factor |
| ECM | Enterprise Content Management |
| EDA | Exploratory Data Analysis |
| EF | Estimation Fulfillment |
| EFT | Estimation Fulfillment Theory |
| EM | Effort Multiplier |
| EP | Explanation and Prediction |
| EPC | Energy, Procurement and Construction |
| ERP | Enterprise Resource Planning |
| ETRM | Energy Trading and Risk Management |
| FFP | Full Function Point |
| FLEX | Development Flexibility *(SF; COCOMO II)* |
| FP | Function Point |

| | |
|---|---|
| FPA | Function Point Analysis |
| FSP | Full Time Equivalent Software Personnel |
| GDP | Gross Domestic Product |
| GUI | Graphical User Interface |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| IBM | International Business Machines (Corporation) |
| ICASE | Integrated Computer-Aided Software Engineering |
| ID | Identifier |
| IDD | Iterative and Incremental Development |
| IEC | International Electro-technical Commission |
| IEEE | Institute of Electrical and Electronics Engineers |
| IFPUG | International Function Point Users Group |
| Inc. | Incorporated |
| IP | Internet Protocol |
| IS | Information System |
| ISO | International Organization for Standardization |
| IT | Information Technology |
| J2EE | Java 2 Platform, Enterprise Edition |
| KDSI | Thousand (Kilo) Delivered Source Instructions |
| KLOC | Thousand (Kilo) Lines of Code |
| KSLOC | Thousand (Kilo) Source Lines of Code |
| LOC | Lines of Code |
| LSS | Logical Source Statements |
| LTEX | Language and Tool Experience *(EM; COCOMO)* |
| MDD | Model-Driven Development |
| MIS | Management Information Systems |
| MM | Man-Month |
| MMRE | Mean Magnitude of Relative Error |
| MS | Microsoft (Corporation) |
| MS-DOS | Microsoft Disk Operating System |
| NASA | National Aeronautics and Space Administration |
| NATO | North Atlantic Treaty Organization |
| NN | Neural Network |
| NS | Nominal Schedule |
| OECD | Organization for Economic Cooperation and Development |
| OLS | Ordinary Least Squares |
| OOD&A | Object-Oriented Design and Analysis |
| PCAP | Programmer Capability *(EM; COCOMO)* |
| PCON | Personnel Continuity *(EM; COCOMO)* |
| PERT | Program Evaluation and Review Technique |
| PF | Productivity Factor |
| PHP | Hypertext Preprocessor |
| PIM | Personal Information Management |
| PLEX | Platform Experience *(EM; COCOMO)* |
| PMAT | Process Maturity *(SF; COCOMO II)* |

| | |
|---|---|
| PREC | Precedentedness *(SF; COCOMO II)* |
| PVOL | Platform Volatility *(EM; COCOMO)* |
| QSM | Quantitative Software Management |
| R&D | Research and Development |
| RELY | Required Software Reliability *(EM; COCOMO)* |
| RESL | Risk Resolution *(SF; COCOMO II)* |
| RUSE | Planned Reusability *(EM; COCOMO)* |
| SAAS | Software as a Service |
| SCED | Required Development Schedule *(EM; COCOMO)* |
| SE | Standard Error |
| SF | Scale Factor |
| SITE | Multisite Development *(EM; COCOMO)* |
| SLIM | Software Life-Cycle Model |
| SLOC | Source Lines of Code |
| SPR | Software Productivity Research |
| SQL | Structured Query Language |
| STOR | Main Storage Constraint *(EM; COCOMO)* |
| SW-CMM | Capability Maturity Model for Software |
| TCF | Technical Complexity Factor |
| TDEV | Development Schedule *("Time to develop")* |
| TEAM | Team Cohesion *(SF; COCOMO II)* |
| TIME | Execution Time Constraint *(EM; COCOMO)* |
| TOOL | Use of Software Tools *(EM; COCOMO)* |
| UAW | Unadjusted Actor Weight |
| UCP | Use Case Point |
| UFP | Unadjusted Function Point |
| UML | Unified Modeling Language |
| UUCP | Unadjusted Use Case Point |
| UUCW | Unadjusted Use Case Weight |
| US | United States |
| UTC | Coordinated Universal Time |
| VAF | Value Adjustment Factor |
| WBS | Work Breakdown Structure |

# 1 EXPOSITION

## 1.1 Motivation

*"You can manage quality into a software product."*[1] According to ROBERT GLASS, this is not a fact but a fallacy. Quality[2] is *not* a management job. Yet, managers can contribute to achieve and ensure quality:

> *"[Managers] can establish a culture in which the task of achieving quality is given high priority. They can remove barriers that prevent technologists from instituting quality. They can hire quality people, by far the best way of achieving product quality. And they can get out of the way of those quality people, once the barriers are down and the culture is established, and let them do what they have wanted to do all long – build something they can be proud of."*[3]

Accordingly, good software is exclusively created by good developers.[4] Developers always take the final steps in development processes. While an appropriate management of software development is *sufficient* in order to facilitate good software, the talents, capabilities, and experiences of developers represent the *necessary* condition:

---

1 GLASS (2002), p. 158.

2 Concerning software, the term *"quality"* has been defined differently in the past (IEEE COMPUTER SOCIETY (2008a)): CROSBY (1979), p. 15, defined quality as the *"conformance to requirements,"* whereas Humphrey (1989), proposed *"achieving excellent levels of fitness for use."* Recently, quality has been defined as *"the degree to which a set of inherent characteristics fulfils requirements"* (ISO (2008)). An overview of the most common quality definitions is given by Jones (2008), p. 455. Alternatively, software quality can be described as follows: High quality software is simply "good software." Each of us might have experienced good or bad software. While we not necessarily recognize good software, we keep in mind the bad things, no matter how marginal they are. A lack of software quality causes dissatisfaction. Let it be technical formatting problems of word-processing software, missing multi-user support of bibliography management software, or synchronization problems with PIM (Personal Information Management) software. Everyone who has ever spent half the day with manually deleting duplicate contacts and appointments, caused by faulty synchronization tools, has developed a personal, but yet appropriate understanding of software quality.

3 GLASS (2002), p. 159.

4 BOEHM (1981); DEMARCO, LISTER (1987); GLASS (2002).

*"The most important factor in software work is not the tools and techniques used by programmers, but rather the quality of the programmers themselves. [...] People matter in building software. That's the message of this particular fact. [...] And yet we keep behaving as if it were not true. Perhaps it's because people are a harder problem to address than tools, techniques, and process."*[5]

In order to support this thought, GLASS provides a set of sources, including DEMARCO and LISTER, who highlight that *"the major problems of our work are not so much technological as sociological in nature,"* as well as DAVIS stating that *"people are the key to success – highly skilled people with appropriate experience, talent, and training are [the] key. The right people with insufficient tools, languages, and process will succeed. The wrong people with appropriate tools, languages, and process will probably fail,"* followed by HIGHSMITH, who encouraged to *"peel back the facade of rigorous methodology projects and ask why the project was successful, and the answer [will be] people,"* and, finally, RUBEY, who wrote, *"when all is said and done, the ultimate factor in software productivity is the capability of the individual software practitioner."*[6]

Accordingly, developers have a major impact on the success of software projects. REEL discusses a set of success factors of which one is having the right team.[7] *"Building the right team means getting good people."*[8] Economists would ask about the productivity of developers. However, in the context of software development, productivity is difficult – if not impossible – to measure.[9] While it is uncomplicated to quantify the input, which is the invested development time, it is difficult to capture or quantify the output. Software is intangible and lacks physical dimensions. Besides, a software system is developed *once*. Consequently, it is senseless to ask about how many units of software *"X"* have been developed during a time interval *"Y"* in order to measure and compare productivity. However, developers as well as projects perform differently.[10]

In view of that, why is one project successful while the other is not? Have all unsuccessful projects been cancelled? How has previous research addressed this issue? Software projects become measureable, quantifiable, and traceable by *estimates*. Effort estimation bridges the gap between managers and the invisible and almost artis-

5 GLASS (2002), p. 11.
6 GLASS (2002), p. 13; DEMARCO, LISTER (1987), p. 4; DAVIS (1994), p. 96; HIGHSMITH (2002), p. 90; RUBEY (1978).
7 REEL (1999).
8 REEL (1999), p. 20.
9 The measurement of productivity is discussed in Section 3.1.3.
10 For a detailed discussion, see Section 3.1.3.1.

tic domain of developers. Estimated and actual efforts provide a *relative* comparison. Projects that were completed with less effort than estimated or with small effort overruns of, for example, 10% might be labeled successful, while projects that exceeded the estimates by 100% or more are usually labeled unsuccessful.

Accordingly, effort estimates are of utmost importance. They give a means to managers to track and control projects. They determine project costs as well as the total customer price. However, if software development is rather sociological in nature, and if people represent a key factor, what happens when developers are confronted with estimates representing the major instrument of management control? Do estimates influence developers, or are they unaffected? Is it irrational to expect that developers start to communicate and discuss estimates, conform to them, work strategically, hide progress or delay, take their time or rush if necessary? Similarly, do project leaders behave as if they watch a mechanical, automated process in which they cannot intervene until completion?

The question whether or not actual project effort is *independent* of estimated effort must be asked. Correspondingly, McConnell states that *"an estimate is a prediction of how long a project will take or how much it will cost. But estimation on software projects interplays with business targets, commitments, and control."*[11] Accordingly, this thesis addresses the posed questions by analyzing effort estimation in general as well as the relationship of estimated and actual effort in particular.

## 1.2 Problem Statement and Purpose of Study

Effort estimation is an essential issue in software development projects. Estimates influence the project planning and the allocation of resources. They determine how much the customer has to pay and whether a company wins or loses a bidding process.[12] Moreover, estimates affect the overall success of a project. Therefore, effort estimation is relevant for both practitioners and researchers. Numerous estimation approaches have been developed over the past decades, starting with Allan Albrecht's *Function Point Analysis* in the late 1970s.[13] Besides the development of new estimation approaches, the evolution and advancement of established approach-

[11] McConnell (2006), p. 3.

[12] Bidding processes, the impact of optimism, and the *Winner's Curse* are briefly discussed in Section 3.1.3.3

[13] Albrecht (1979). See Section 2.2.2.

es, as well as the abandonment of approaches that never prevailed, effort estimation techniques have constantly been subject to research on *estimation accuracy*.[14]

Accuracy is the central property of estimation techniques. It represents the goodness of an estimation technique. Actual project effort can be underestimated, overestimated, or perfectly predicted. Underestimated projects – unless deliberately underestimated due to strategic reasons – are problematic, as they lead to effort overruns, and, consequently, to less profit or even financial loss. Yet, overestimation is problematic as well, since it might hinder the successful acquisition of new projects. Analogous to the number of available estimation approaches, there are numerous *accuracy measures*. Additionally, there are debates about the usefulness, applicability, and comparability of such measures.[15]

This thesis neither aims at the development of yet another estimation approach, nor does it address the development or improvement of accuracy measures. Instead, this thesis focuses on the general nature of estimates and on how they are utilized in software development organizations. Estimation approaches as well as the relationship between actual and estimated effort are generally described, discussed, and subject to research studies as if actual and estimated effort were independent measures.[16] However, there is doubt that the assumption of independence always holds true. Accordingly,

**Research question 1:** *Is it reasonable to assume a general independency of estimated and actual software development effort?*

The purpose of this thesis is to build a *theory* that emerges from this research question. Software development projects are not understood as purely *technical* processes that are determined and carried out by formal process definitions, rigorous planning, perfect specifications, routine work, as well as objective and non-influencing estimates. Instead, software development projects are understood as creative, team-based, less formal, as well as highly communicative processes, which establish the setting for the innovative and non-routine work of developers. As long as people matter in building software, development projects are *sociological* in nature. The

[14] E.g., FINNIE *et al.* (1997); FROHNHOFF (2008); MUKHOPADHYAY *et al.* (1992); SHEPPERD, SCHOFIELD (1997).

[15] KITCHENHAM *et al.* (2001).

[16] E.g., FROHNHOFF (2008) focuses on estimation accuracy by comparing two estimation approaches. The author highlights the importance of accurate measures without doubting the independence of actual and estimates effort. There is only little research that explicitly addressed the potential impact of estimates on actual project performance, e.g., ABDEL-HAMID, MADNICK (1991); ABDEL-HAMID, MADNICK (1986); BOEHM (1981); JORGENSEN, SJOBERG (2001); MCCONNELL (2006); SOMMERVILLE (2006). An in-depth discussion of this issue is given in Section 5.4.

theory construction takes this idea into account and addresses the interplay of project lead and developers throughout a project. The central goal of the theory is to *explain* and to *predict how* estimates influence project lead and developers, *how* developers influence estimates, and *why* software development projects miss their target values under specific circumstances. Correspondingly,

**Research question 2:** *How do developers and project lead influence the relationship between estimated and actual software development effort?*

## 1.3 Research Approach and Methodology

In the field of Information Systems (IS) research, there are ongoing discussions and academic debates on research methodology.[17] These debates are often driven by different, partially conflicting philosophical convictions. One debate, for example, addresses the controversy of *rigor* and *relevance* in IS research.[18] Another major debate focuses on *interpretivism* vs. *positivism*.[19] Commonly, these debates emerge from different *epistemological* and *ontological* positions.[20] Although positions are different and often regarded as mutually exclusive[21], they all aim at ensuring rigorous research.

The questions concerning the existence of a real world (ontological perspective) as well as the relationship between cognition and the object of cognition (epistemological perspective) have been intensively discussed in the IS literature.[22] Different ontological and epistemological positions are essential to the debate of positivism and interpretivism.[23] LEE developed a framework that integrates both interpretivism and positivism.[24] This framework incorporates *three levels of understanding* (see Fig. 1.1).

[17] E.g., GREGOR (2006); HEVNER *et al.* (2004).
[18] LEE (1999).
[19] LEE (1991).
[20] E.g., HOLTEN *et al.* (2004).
[21] E.g., BURRELL, MORGAN (1979).
[22] E.g., BECKER, NIEHAVES (2007); CHEN, HIRSCHHEIM (2004); FALCONER, MACKAY (1999); FITZGERALD, HOWCROFT (1998); FITZGERALD *et al.* (1985); HIRSCHHEIM (1985); LEE (1991); WALSHAM (1995); WEBER (2004).
[23] E.g., BECKER, NIEHAVES (2007); HOLTEN *et al.* (2004); LEE (1991); WEBER (2004).
[24] LEE (1991).

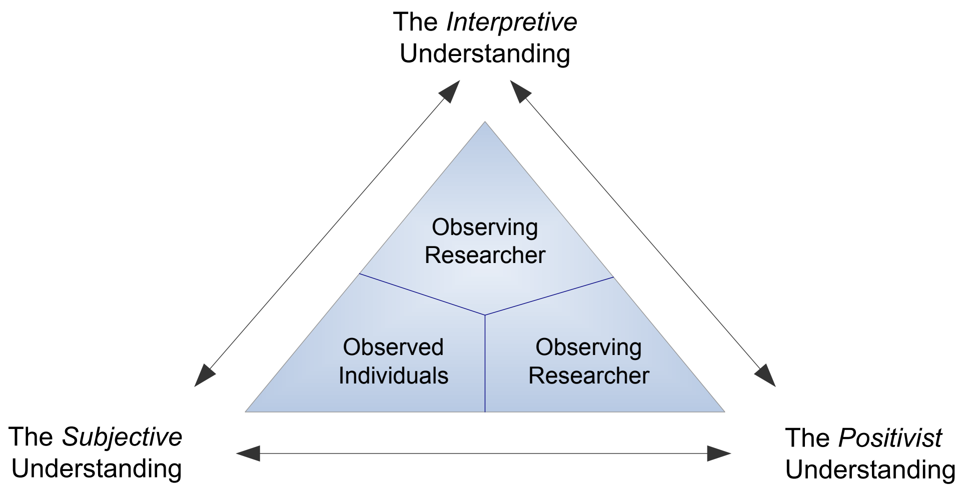

The *Interpretive* Understanding
Observing Researcher
Observed Individuals
Observing Researcher
The *Subjective* Understanding
The *Positivist* Understanding

**Fig. 1.1:** Cyclical nature of the three levels of understanding[25]

First, the *subjective understanding* belongs to the observed individuals. This level of understanding refers to common sense and everyday meanings, which determine how individuals perceive themselves and how they behave in particular socially constructed settings. Second, the *interpretive understanding* belongs to the observing researcher. This understanding results from the observation, perception, and interpretation of the subjective, common-sense understanding. Third, the *positivist understanding* also belongs to the observing researcher who created this level of understanding in order to *explain* the investigated empirical reality. An explanation or scientific theory, respectively, is based on constructs that belong exclusively to the researcher. Moreover,

> *"The explanation consists of formal propositions that typically posit the existence of unobservable entities (like social structure). [...] In taking the form of formal propositions and in referring to factors unknown to the human subjects themselves, the theoretical explanation at the third [positivist] level is qualitatively different from both the common-sense understanding located at the first [subjective] level and the interpretation of the common-sense understanding located at the second [interpretive] level. In addition, the theoretical explanation located at the third level must obey the same rules of formal logic and controlled empirical testing that apply to scientific explanations in general."*[26]

Accordingly, the subjective understanding represents the basis from which the interpretive understanding emerges. The interpretive understanding is the basis used to

[25] LEE (1991).
[26] LEE (1991), p. 351.

develop the positivist understanding by generating propositions or hypotheses, as well as confronting and testing them with empirical content.[27]

Regarding software development projects, it is assumed that the process of building software takes place in a "*real*" world that exists independently of cognition, thought, and speech (*ontological realism*; see Fig. 1.2).[28] Yet, the observation of software development processes is regarded as "*subjective*" or "*private*" for all participants, i.e., managers, developers, as well as observing researchers (*subjective cognition*; see Fig. 1.2).[29] Concerning the relationship between cognition and the object of cognition, *"the continuum ranges from the assumption that the cognition of an objective reality is interpreted by the subject [interpretivism] to the assumption that cognition is private, because of the idea that there does not exist such thing as an objective reality [radical constructivism]."*[30]

| | | **Epistemological position** | |
|---|---|---|---|
| | | Objective cognition is impossible (subjective cognition) | Objective cognition is possible (epistemological realism) |
| **Ontological position** | A real world is existent (ontological realism) | **Interpretivism** | **Positivism** |
| | No real world exists (idealism) | **Radical constructivism** | |

**Fig. 1.2:** Ontological and epistemological positions[31]

This thesis assumes that there is both an *objective* and a *subjective* reality in software development projects, and that both realities are relevant for investigating and understanding phenomena of software development. With reference to the necessity to integrate objective and subjective realities in IS research, RECKER gives an illustrative example:

> *"A process modeling initiative may succeed or fail based on whether the models were completed on-time and on-budget (which would be factual, objective reality). However, the different stakeholders involved in the initiative may still have their own perceptions on whether that particular initiative was a success or failure and why (which would be a subjective reality). Each of these view-*

[27] E.g., ROSENKRANZ, HOLTEN (2007).
[28] E.g., BECKER, NIEHAVES (2007); HOLTEN *et al.* (2004).
[29] E.g., BECKER, NIEHAVES (2007); HOLTEN *et al.* (2004).
[30] BECKER, NIEHAVES (2007), p. 203.
[31] C.p. HOLTEN *et al.* (2004), p. 178; cp. NIEHAVES *et al.* (2004), p. 4234.

> *points provides its own lens through which a phenomenon can be investigated, and both can contribute to understanding. In fact, a number of researches have lamented that relying on just one method, theory, or methodology would lead to an inconsistency between theory and actual practice (i.e., there is more to IS practice than IS theories can explain)."*[32]

This thesis affirms the necessity to take both realities into account, and, in consequence, its research design makes use of interpretive as well as positivist approaches. Due to the given combination of ontological and epistemological assumptions, this thesis belongs to the position of *interpretivism*. However, with reference to LEE'S suggestions on the integration of interpretivism and positivism, an interpretive position is not contradictory to theory construction.

Following GREGOR, theory construction should address (at least) one of the following four primary goals: 1) *Analysis and Description*, 2) *Explanation*, 3) *Prediction*, or 4) *Prescription*.[33] The goals of explanation and prediction (EP) can be combined. Accordingly, GREGOR derives five theory types, which are briefly outlined in Tab. 1.1.

**Tab. 1.1:** Taxonomy of theory types in information systems research[34]

| Theory Type | | Distinguishing Attributes |
|---|---|---|
| I. | Analysis | *Says what is.*<br>The theory does not extend beyond analysis and description. No causal relationships among phenomena are specified and no predictions are made. |
| II. | Explanation | *Says what is, how, why, when, and where.*<br>The theory provides explanations, but does not aim to predict with any precision. There are no testable propositions. |
| III. | Prediction | *Says what is and what will be.*<br>The theory provides predictions and has testable propositions, but does not have well-developed justificatory causal explanations. |
| IV. | Explanation and prediction | *Says what is, how, why, when, where, and what will be.*<br>The theory provides predictions and has both testable propositions and causal explanations. |
| V. | Design and action | *Says how to do something.*<br>The theory gives explicit prescriptions (e.g., methods, techniques, principles of form and function) for constructing an artifact. |

This thesis aims at the development of a so-called *EP Theory* which focuses on both explanation and prediction: *"EP theory implies both understanding of underlying causes and prediction, as well as description of theoretical constructs and the rela-*

32 RECKER (2008), p. 25.
33 GREGOR (2006).
34 GREGOR (2006), p. 620.

*tionships among them.“*[35] In consequence, the theory will describe the relevant constructs, explain their relationships, provide a causal model as well as testable propositions in order to answer all relevant *"what is, how, why, when, and what will be"* questions.[36] As a second step, the theory will be tested by confronting it with empirical content based on expert interviews conducted with software professionals.

## 1.4 Research Design

***Exploratory study***

This research begins with an *exploratory study*.[37] This type of study is helpful when not much is known about a particular situation or phenomenon.[38] While the techniques of effort estimation are regarded as well studied and extensively documented[39], and while the deviation of estimated and actual effort has been subject to previous research[40], there is no information available that addresses the distribution of effort invested by individuals when no effort estimates are given. This particular phenomenon is the central subject of the exploratory study. Above and beyond, the study's motivation is to identify interesting or controversial causal relationships in the data, as well as getting an inspiration for a potential theory development from its analysis.

The exploratory study represents the first phase of this research. It is based on empirical data gained from an e-learning platform nicknamed *"Playgrounds,"* on which students learn and practice SQL and Python programming.[41] The platform provides several e-learning applications with different focuses. Two datasets were generated from more than 150,000 SQL and Python codes, written and executed over two semesters. The major part of the exploratory study uses a particular data subset that was gained from sessions of one e-learning application, called *"SQL Trainer."* After data cleansing, this data subset contains 16,577 SQL queries that were executed by 170 second semester bachelor students.

---

[35] GREGOR (2006), p. 626.
[36] GREGOR (2006), p. 626.
[37] See Chapter 4.
[38] E.g., SEKARAN (2003), BLESS *et al.* (2007), KOTLER *et al.* (2006), STEINBERG, STEINBERG (2005), BLANCHE *et al.* (2008), BORTZ, DÖRING (2006). For a detailed discussion of exploratory studies as a research method, see Section 4.1.
[39] See Chapter 2.
[40] See Section 2.4 as well as Section 3.1.3.
[41] *SQL (abbr.): Structured Query Language.*

The study concentrates on a *quantitative* analysis based on the given dataset. Since this phase is exploratory, it is not designed to test hypotheses but to develop insights, assumptions, and/or propositions. With reference to the importance of individual differences between software developers as well as the doubted independence of actual and estimated effort, the method of *Simple Random Sampling* is used in order to generate small simulated development projects that are then subjected to different statistical analyses. The result of each data analysis is interpreted. These interpretations largely concentrate on how well development effort can be estimated.

***Theory Development***

The second phase of this research focuses on the construction of a new theory that offers both *explanation* and *prediction* (EP-type).[42] At first, a *causal model* is developed that predominantly explains the interplay of managers and developers in software development projects. This causal model incorporates the interests and behavior of project leaders, the interests and behavior of developers, the influence of individual capabilities of developers, as well as the impact of the inherent properties of software systems.[43] The developed causal relationships represent the *explanatory* component of the EP theory (*"what is, why, and how"*).[44]

Based on the causal model, the central *theory statement* and two sets of *assumptions* and *propositions* are developed. The assumptions describe under which circumstances a particular phenomenon can be observed. The set of propositions describes the implications of this phenomenon. The assumptions (*"when and where"*) and propositions (*"what will be"*) represent the *predictive* component of the EP theory.[45]

***Theory Validation***

The third phase addresses the validation of the developed EP theory.[46] The validation is divided into two steps. First, the propositions are tested by *semi-structured expert interviews* (qualitative inquiry).[47] Accordingly, professional software developers, who have relevant project experience and who are concerned with effort estimation, were contacted and chosen as interviewees. If possible, the heads of development were chosen as interview partners. Based on their responses, it can be determined whether or not the assumptions and the propositions of the theory are supported. The goal of this first step is to test if the interview responses *corroborate* the theory, or if

[42] GREGOR (2006). See Chapter 5.
[43] BROOKS (1975). See Section 5.1.4.
[44] C.p. GREGOR (2006). See Section 5.1.
[45] C.p. GREGOR (2006). See Section 5.2.
[46] See Chapter 6.
[47] FONTANA, FREY (2000); MYERS, NEWMAN (2007). See Section 6.1.1.

the interview responses lead to a *falsification*.[48] Besides, the responses are analyzed as to whether they imply *counter-findings*, which do not falsify the theory in general, but which provide reasonable and comprehensible *limitations*.[49]

Finally, the process and the result of the theory construction, including its causal model, its assumptions, its propositions, as well as the conclusion taken during theory development, is subject to an *expert review* conducted with one professional software developer and project manager.[50] The goal of the expert review is to determine whether or not practitioners can comprehend and agree upon the theory construction. In addition, it can be discussed whether the findings from the expert interviews appear to be correctly understood, and whether they lead to acceptable conclusions from the practitioner's perspective.

***Integration of Interpretivism and Positivism***

During this thesis, all three levels of understanding, i.e., the *subjective*, the *interpretive*, and the *positivist* understanding, are addressed. With reference to the framework suggested by LEE, the following Fig. 1.3 visualizes the stepwise interplay of interpretivism and positivism in this research.

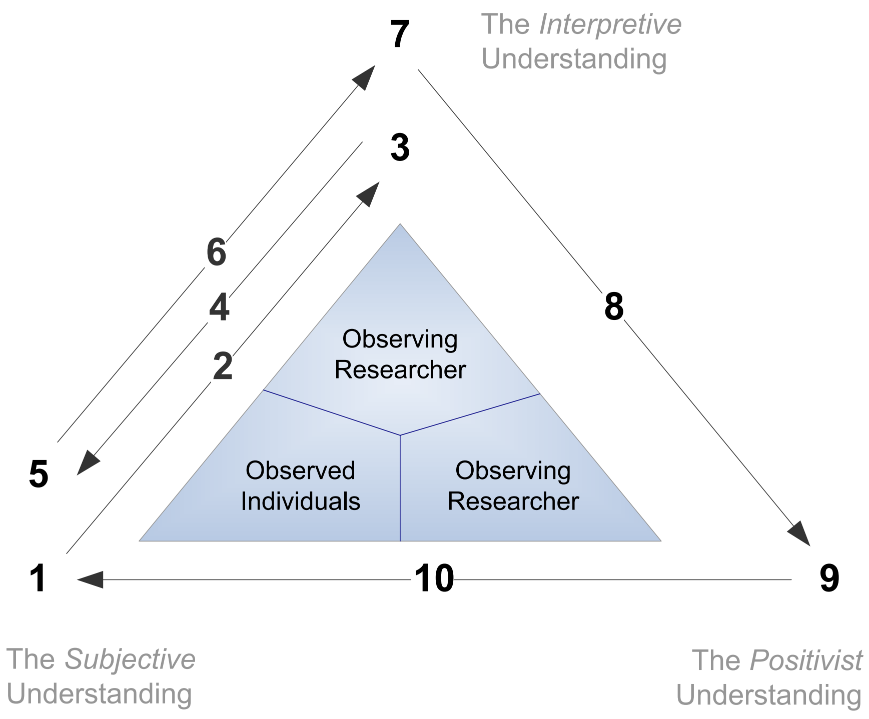

7
The Interpretive Understanding
3
6
4
2
8
Observing Researcher
Observed Individuals
Observing Researcher
5
1
10
9
The Subjective Understanding
The Positivist Understanding

**Fig. 1.3:** Steps of addressing the three levels of understanding during research

[48] LEE (1991); POPPER (1959). See Section 6.1.2.
[49] KUHN (1970).
[50] See Section 6.4.

Step 1 represents the initial subjective knowledge and points of view of the researcher. This knowledge is influenced by personal experiences with software development projects, academic education, literature, as well as previous research. Moreover, this knowledge was subjected to interpretation before this research has been started (Step 2). The developed interpretive understanding (Step 3) has an impact on both the design and focus of the exploratory study (Step 4).

The exploratory phase consists of two steps. Step 5 addresses the *construction of data*, i.e., the particular setting by which the data is gained. Accordingly, this step belongs to the subjective understanding.[51] With respect to the considerations on generalizability of LEE and BASKERVILLE, the exploratory study

> *"[...] generalizes from empirical statements (as inputs to generalizing) to other empirical statements (as outputs of generalizing). [...] The generalizability of data to a measurement, observation, or other description (such as a descriptive statistic or a thick description) and the generalizability of the resulting measurement, observation, or other description beyond the sample or domain from which the researcher has actually collected data [...]."*[52]

The construction of data addresses the *"first-level constructs [that] refer to the understandings held by the observed people themselves."*[53] LEE argues that

> *"The subjective understanding provides the basis on which to develop the interpretive understanding."*[54]

Analyses based on the data are "*second-level constructs [that] refer to the understanding held by the observing researcher."*[55] Step 5, i.e., the construction of data, is the foundation for Step 6, which addresses the analysis of this data. Step 6 makes use of *quantitative* methods in order to explore the field. While the data, understood as a set of values, is regarded as *objective* and *precise*, since it is exclusively based on time measures, its perception, analysis, and interpretation is still regarded as *subjective*. Hence, Step 6 belongs to the interpretive understanding:

> *"This understanding is the researcher's reading or interpretation of the first-level common-sense understanding."*[56]

---

51 ROSENKRANZ, HOLTEN (2007).
52 LEE, BASKERVILLE (2003), pp. 232-233.
53 LEE, BASKERVILLE (2003), p. 230.
54 LEE (1991), p. 351.
55 LEE, BASKERVILLE (2003), p. 230.
56 LEE (1991), p. 351.

In consequence, the interpretation and analysis of the data contribute to and influence the existing interpretive understanding (Step 7). This level of understanding is the basis for developing the *EP theory* (Step 8). Since the theory has a *predictive* component, consisting of *propositions* that can be tested by empirical, quantitative data, the theory belongs to the positivist understanding (Step 9):

> *"[The positivist] understanding is one that the researcher creates and tests in order to explain the empirical reality that he or she is investigating. This explanation, which is also called scientific theory, is made up of constructs that belong exclusively to the researcher (as opposed to the observed human subjects). The explanation consists of formal propositions that typically posit the existence of unobservable entities (like social structure). [...] In addition, the theoretical explanation located at the third level must obey the same rules of formal logic and controlled empirical testing that apply to scientific explanations in general."*[57]

Finally, the *validation* of the theory, by conducting semi-structured expert interviews, confronts the positivist understanding, i.e., the theory, with private experiences of the observed individuals, and, accordingly, with their subjective understandings (Step 10):

> *"The subjective understanding, through the publicly observable behaviors arising from it, may serve to confirm or disconfirm the predictions of the positivist understanding. [...] The theoretical propositions must satisfy the rules of formal logic, the rules of hypothetico-deductive logic, and the four requirements of positivism, falsifiability, logical consistency, relative explanatory power, and survival."*[58]

## 1.5 Organization of Thesis

Following this introduction, *Chapter 2* focuses on the theoretical foundations of this thesis. The second chapter starts with a brief discussion of *Software Engineering* and its relation to *Software Development* (Section 2.1). Before turning to *Effort Estimation* as the central subject of this thesis, relevant concepts of *Software Sizing*, including common size measures (Section 2.2.1), the technique of *Backfiring* (Section 2.2.2.2), as well as *Function Points Analysis* (Section 2.2.2) and its variants (Section 2.2.3), are introduced. Section 2.3 focuses on *Effort Estimation*. Each discussed estimation technique is classified as either being *model-based* (Section 2.3.1), *expertise-*

[57] LEE (1991), p. 351.
[58] LEE (1991), p. 353.

*based* (Section 2.3.2), *learning-oriented* (Section 2.3.3), or *dynamics-based* (Section 2.3.4). Section 2.4 gives a brief summary and discussion of the most common estimation techniques.

*Chapter 3* addresses software development from an *economic* perspective. For that reason, the emergence and the evolution of the so-called *Software Crisis* are discussed (Section 3.1). This discussion focuses on the *reliability*, *control*, and *cost-effectiveness* of software development. Section 3.1.2 discusses the evolution as well as the possibilities and limitations of managing software projects. During this discussion, different eras of software development, i.e., *sequential*, *iterative*, and *agile* software development, are presented. Section 3.1.3 addresses the aspect of cost-effectiveness, and, therefore, puts a strong emphasis on the economic perspective. Among other issues, this section discusses the myth of the Chaos Report, the lead of traditional engineering over software engineering, as well as the measurement of productivity of software development. The corresponding findings are the basis for elaborating the *crux* of software development (Section 3.3). Before, the production of software is opposed to traditional mass production and its special economies, i.e., *economies of scale* and *economies of scope* (Section 3.2). This section aims at characterizing the nature of software development and explaining why software development is generally different from traditional production. Finally, the implications on effort estimation are described (Section 3.3).

*Chapter 4* describes the conducted *exploratory study*. First, the student datasets and the corresponding *data retrieval* processes during the summer semester of 2008 and the winter semester of 2008/09 are explained (Section 4.2). The main part of Chapter 4 addresses the exploration of one particular data subset (Section 4.3). The data analysis concentrates on the *solution capability* (Section 4.3.1), the overall *solution effort* (Section 4.3.2), as well as the individual *working speed* of students (Section 4.3.3). Afterwards, the concept of *Simple Random Sampling* is introduced, which is required to simulate small development projects (Section 4.3.4). The analysis of estimation accuracy is introduced in Section 4.3.5. Section 4.3.6 incorporates the PARKINSON effect into the random sampling process. In following analyses, different intensities of the PARKINSON effect as well as simple forms of *project control* are tested (sections 4.3.6 and 4.3.7). Section 4.4 summarizes the gained insights, observed phenomena, and drawn conclusions based on the exploratory study.

*Chapter 5* addresses the development of the EP theory. Section 5.1 describes the *causal model*, including the *role and interests of project lead* (Section 5.1.1), the *role and interests of developers* (Section 5.1.2), the *craftsmanship of skilled individuals*

(Section 5.1.3), as well as the impact of *accidental complexity* on effort (Section 5.1.4). Afterwards, Section 5.2 presents the central theory statement, its assumptions, and its propositions. Section 5.3 describes the empirical observability of the theory and introduces the concept of *estimation fingerprints*. Subsequent to the presentation of the EP theory, the most closely related research is discussed in Section 5.4.

*Chapter 6* reports on the theory validation. Section 6.1 focuses on the *interview design* (Section 6.1.1) and the qualitative *research method* used to analyze the expert responses (Section 6.1.2). Biographical information about the experts and the most important answers and statements given during the interviews can be found in Section 6.2. Next, Section 6.3 discusses the *findings* gained from the interviews. Section 6.4 presents the *expert review*, including the review of the causal model (Section 6.4.1), the review of the assumptions and the propositions as well as the review of the central theory statement and the estimation fingerprints (Section 6.4.2).

*Chapter 7* concludes this thesis. At first, the research process is summarized (Section 7.1). Afterwards, Section 7.2 discusses the major contributions of this research in brief. Section 7.3 presents the implications for research and practice, followed by Section 7.4 reporting on limitations. At last, Section 7.5 suggests future research.

Fig. 1.4 visualizes the corresponding thesis structure.

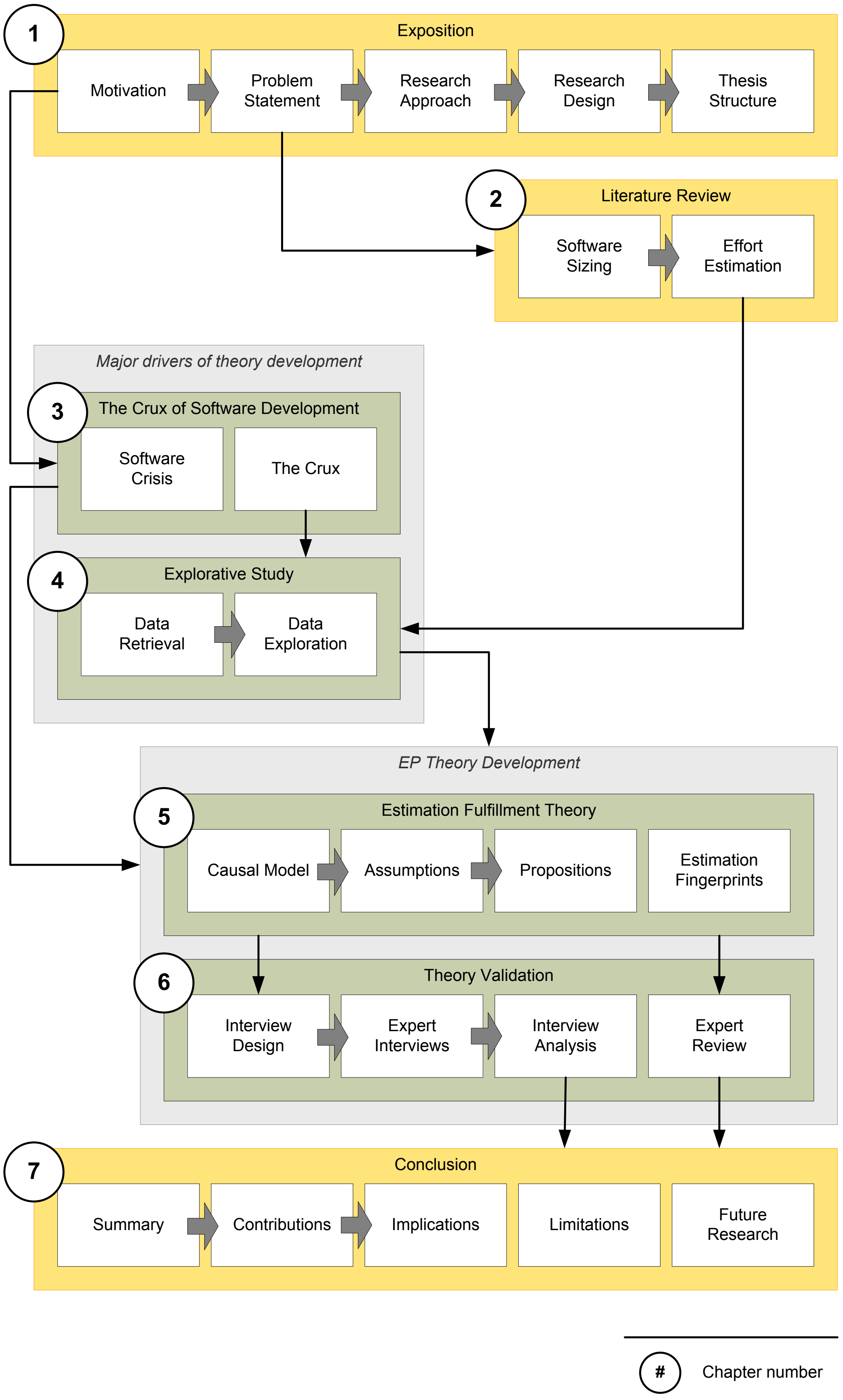


**Fig. 1.4:** Thesis structure

# 2 LITERATURE REVIEW

## 2.1 Software Engineering

This thesis focuses on problems, challenges, and research questions as well as the history and state-of-the-art of *software development* and *software engineering*. Before turning to the review of relevant literature, both terms - "*software development*" and "*software engineering*" - are briefly discussed and distinguished from each other.

The term "software engineering" has been introduced to the public by the "*NATO Conference of Software Engineering"* which was held in Germany in 1968.[59] Chairman of this conference was FRITZ L. BAUER who initially coined the term.[60] BRIAN RANDELL, who was one of the two conference editors, remembers:

> *"The idea for the first NATO Software Engineering Conference and in particular that of adopting the then practically unknown term 'software engineering' as its (deliberately provocative) title, I believe came originally from Professor FRITZ BAUER. Similarly, if my memory serves me correctly, it was he who stressed the importance of providing a report on the conference and who persuaded PETER NAUR and me to be the editors."*[61]

Recalling the first two conferences on software engineering in 1968 and 1969, RANDELL describes how the discipline was perceived at that time:

> *"Unlike the first conference, at which it was fully accepted that the term software engineering expressed a need rather than a reality, in Rome there was already a slight tendency to talk as if the subject already existed."*[62]

59 NAUR, RANDELL (1969), p. 1027. *NATO (abbr.): North Atlantic Treaty Organization.*
60 NAUR, RANDELL (1969).
61 RANDELL (1996), p. 37.
62 RANDELL (1996), p. 41.

This quotation documents that having a name for a new discipline in the 1960s was much more important than working on clear definitions of software engineering. Furthermore, the term was easy to comprehend due to the existence of many other engineering disciplines, like *civil*, *mechanical*, or *chemical engineering*. The OXFORD ENGLISH DICTIONARY defines *Engineering* as *"The work done by, or the profession of, an engineer"* as well as *"The art and science of the engineer's profession."*[63] Moreover, *Engineer* is defined as *"One who contrives, designs, or invents."*[64] In view of that, the definition of *Engineer* and, therefore, the definition of *Engineering* are applicable for software. Additionally, the OXFORD ENGLISH DICTIONARY defines *Software Engineering* as *"the professional development, production, and management of system software."*

A more detailed definition, for example, is given by the IEEE COMPUTER SOCIETY[65], which defines *Software Engineering* as:

> *"1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. 2) The study of approaches as in 1)."*[66]

In textbooks on software engineering, the central idea of software engineering is described in more detail, but the quintessence is identical. For example, SOMMERVILLE introduces software engineering as follows:

> *"Software engineering is concerned with the theories, methods, and tools which are needed to develop the software [...]. In most cases, the software systems [...] are large and complex systems. They are also abstract in that they do not have a physical form. Software engineering is therefore different from other engineering disciplines. It is not constrained by materials governed by physical laws or by manufacturing processes."*[67]

Another definition of software engineering is given by SCHACH who puts the emphasis on the outcome of the underlying production process:

---

63 SIMPSON, WEINER (1989).
64 SIMPSON, WEINER (1989).
65 *IEEE (abbr.): Institute of Electrical and Electronics Engineers.*
66 IEEE COMPUTER SOCIETY (2008B); IEEE STANDARDS ASSOCIATION (1990).
67 SOMMERVILLE (1996), p. 4.

> *"Software engineering is a discipline whose aim is the production of fault-free software, delivered on time and within budget, that satisfies the client's needs."*[68]

The definitions of the OXFORD ENGLISH DICTIONARY and the IEEE COMPUTER SOCIETY affirm that software engineering can be rightly regarded as an engineering discipline. Generally, an engineering discipline is both an art and a science, which demands systematic and disciplined work from its engineers. Additionally, the second sentence of the IEEE definition states that studying approaches of software engineering is also part of the discipline.[69] This aspect points at the high demand for self-reflection in the field of software engineering.

Generally, software engineering can be divided into three areas, i.e., *software development*, *operation*, and *maintenance*. This specifies the relation of software engineering to software development. Software development is subordinated to software engineering. Moreover, software development is focused on and limited to the construction of new software products.

The definition of SOMMERVILLE differentiates software engineering from other classic engineering disciplines, since software is *intangible*.[70] The missing physical form makes it difficult to visualize software and to determine its size.

Finally, the definition of SCHACH describes constraints in which software development is embedded in.[71] Limits set by schedules, budgets, and quality standards refer to the *economic* motivation for software engineering, especially forms of systematic and disciplined software development.

## 2.2 Software Size Estimation

In the context of software development, the primary cost driver is *effort*.[72] Consequently, the total costs of a development project largely depend on the invested working time. Before estimating effort, however, the *size* of software must be determined. As stated above, software is *intangible,* and, consequently, it has no physical dimensions that describe its size. Therefore, software engineering must provide techniques for *software sizing* which represents a pre-stage for effort estimation tech-

68 SCHACH (2005).
69 IEEE COMPUTER SOCIETY (2008B); IEEE STANDARDS ASSOCIATION (1990).
70 SOMMERVILLE (1996).
71 SCHACH (2005).
72 BOEHM (1981).

niques. In the following, the most prominent size measures, i.e., *Lines of Code* as well as *Function Points* and its variants, are briefly presented.

### 2.2.1 Code-based Size Measures

*Lines of Code* (LOC) is a quantitative and straightforward *size* measure for software systems. Typically, the size of software is measured in *KLOC*– one thousand Lines of Code.[73] PARK gives a compact overview of the most frequent acronyms in this context:

> *"Some common acronyms for physical source lines are LOC, SLOC, KLOC, KSLOC, and DSLOC, where SLOC stands for source lines of code, K (kilo) indicates that the scale is in thousands, and D says that only delivered source lines are included in the measure. Common abbreviations for logical source statements include LSS, DSI, and KDSI, where DSI stands for delivered source instructions."*[74]

However, for a given set of requirements, Lines of Code vary across different programming languages and with the extent of automatic code generation.[75] Additionally, the definition of SLOC is not standardized. Some counting definitions include empty lines, comments, and brackets while others do not.[76]

According to BOEHM, the best approach to estimate the number of Lines of Code, which are necessary to develop a new software product, is *historical data*.[77] The number of LOC of a completed project is known *ex post* since LOC are objectively measurable once a specific counting definition has been chosen. However, if data of past projects *is* available and if that data allows identifying analogies with the current project, it will be possible to estimate the number of LOC for the new software system or its components *ex ante*.[78] If historical data is *not* available, it will be *still* possible to use expert opinions in order to attain estimates of likely, lowest-likely, and highest-likely size.[79]

[73] BOEHM *et al.* (2000b).
[74] PARK (1992), p. 13.
[75] MATSON *et al.* (1994).
[76] MATSON *et al.* (1994).
[77] BOEHM *et al.* (2000b).
[78] E.g., SHEPPERD, SCHOFIELD (1997); SHEPPERD *et al.* (1996).
[79] BOEHM *et al.* (2000b).

### 2.2.2 Function Points

The *Function Point* (FP) estimation technique or *Function Point Analysis* (FPA) determines the software size by quantifying its information processing functionality.[80] This quantification approach was initially published by ALLAN ALBRECHT in 1979.[81] Function Points are especially helpful for early estimations since the approach works with premature, vague information.[82] FPA was primarily designed for the domain of *Management Information Systems* (MIS), where it has been extensively analyzed and evaluated.[83] Since 1986, FPA is being maintained by the *International Function Point Users Group* (IFPUG), which provides certifications and estimation guidelines for software professionals.[84] FPA is still the most widely used formal estimation technique today.[85]

#### 2.2.2.1 Counting Technique

Basically, the functional user requirements are categorized and counted in terms of so-called *User Function Types*, i.e., External Input, External Output, Internal Logical File, External Interface File, and External Inquiry.[86]

In the context of FPA, *External Input* represents user data or user control input that *enters* the external boundary of the software system. Analogously*, External Output* represents user data or user control output that *leaves* the external boundary. *Internal Logical Files* refer to logical groups of user data or control information that are generated, used, or maintained by the system. *External Interface Files* are passed or shared between software systems. Finally, *External Inquiries* refer to unique input-output combinations, where input causes and generates an immediate output.[87]

Each instance of the categorized requirements is classified by three complexity levels, i.e., *Simple*, *Average*, and *Complex*.[88] Each complexity level refers to a set of weights ranging from 3 to 15 that influence the Function Point counting (see Tab. 2.1). The resulting counting represents the *Unadjusted Function Points* (UFP).[89]

80 BOEHM *et al.* (2000b).
81 ALBRECHT (1979).
82 BOEHM *et al.* (2000b).
83 ABRAN, ROBILLARD (1996); JONES (1996); KEMERER (1987); KEMERER (1993); SYMONS (1988).
84 INTERNATIONAL FUNCTION POINT USERS GROUP (1994).
85 GARMUS, HERRON (2001); JANTZEN (2008).
86 ALBRECHT, GAFFNEY JR (1983).
87 ALBRECHT, GAFFNEY JR (1983).
88 ALBRECHT, GAFFNEY JR (1983).
89 BOEHM *et al.* (2000b).

**Tab. 2.1:** UFP complexity weights[90]

| | Complexity Weight | | |
|---|---|---|---|
| **Function Type** | **Low** | **Average** | **High** |
| External Inputs | 3 | 4 | 6 |
| External Outputs | 4 | 5 | 7 |
| Internal Logical Files | 7 | 10 | 15 |
| External Interface Files | 5 | 7 | 10 |
| External Inquiries | 3 | 4 | 6 |

UFP is used, for example, by the *Constructive Cost Model* (COCOMO) to reflect the software size. A brief introduction of how to count and assess functional user requirements is given by ALBRECHT.[91] Detailed guidelines are given by IFPUG.[92]

ALBRECHT proposes a further step in order to *adjust* the FP counting. After the requirements have been listed, classified and counted, the resulting UFP can be adjusted by taking *general application characteristics* into account.[93] These characteristics reflect the *Processing Complexity* of the software system. ALBRECHT notes 14 general characteristics that influence the FP counting.[94]

For each application characteristic, the *Degree of Influence* (DI) is rated on a scale from 0 (“no influence”) to 5 (“strong influence”).[95] The sum of all fourteen DI determines the *Value Adjustment Factor* (VAF), which ranges from 0.65 to 1.35 allowing an overall adjustment of ± 35%.[96] The *Adjusted Function Points* (AFP) are calculated by multiplying the *Value Adjustment Factor* with the *Unadjusted Function Points*:

$$\begin{aligned} &AFP = UFP \times VAF \\ &VAF = 0.65 + 0.01\sum_{i=1}^{14} DI_i \quad with\ DI_i \in [0;0.05] \end{aligned} \qquad \textbf{Eq. 2.1}$$

The *Function Point Analysis* is often referred to as a technique to estimate *effort*. It has to be pointed out that both UFP and AFP are *size* measures. These measures, however, can be used for estimating effort by applying a company-specific factor

[90] BOEHM *et al.* (2000b)
[91] ALBRECHT, GAFFNEY JR (1983).
[92] INTERNATIONAL FUNCTION POINT USERS GROUP (1994).
[93] ALBRECHT, GAFFNEY JR (1983).
[94] ALBRECHT, GAFFNEY JR (1983); JANTZEN (2008). The 14 general application characteristics are: *data communications, distributed data processing, performance objectives, heavily used equipment, high transaction rates, online data entry, end user efficiency, online updates, complex processing, reusability, installation ease, operational ease, multiple installation sites*, as well as *the ability to facilitate change.*
[95] BOEHM *et al.* (2000b); KEMERER (1987).
[96] ALBRECHT, GAFFNEY JR (1983); BOEHM *et al.* (2000b); JANTZEN (2008).

that transforms Function Points into a time-based effort measure like man-month. This direct transformation assumes a waterfall-oriented development process.[97]

### 2.2.2.2 Backfiring Technique

Based on historical data and regression analyses, it is possible to relate *Unadjusted Function Points* to *Source Lines of Code*.[98] The technique of converting UFP to SLOC – called *Backfiring* – is language-dependent. The conversion ratios for different programming languages are listed in *Backfiring Tables*. The following Tab. 2.1 presents conversion rates for a set of programming languages based on two different historical data sources. With reference to the conversion rates given by JONES (Tab. 2.1), the implementation of one UFP requires, for example, 213 source lines of Assembler code on average, while the same implementation requires 53 source lines when programmed in Java.

**Tab. 2.2:** UFP to SLOC conversion ratios[99]

| | **SLOC/UFP** | |
|---|---|---|
| **Language** | **Jones (1996)**[100] | **QSM (2005)**[101] |
| 1. Generation Language | 320 | - |
| 2. Generation Language | 107 | - |
| 3. Generation Language | 80 | - |
| 4. Generation Language | 20 | - |
| 5. Generation Language | 4 | - |
| Assembler | 213 | 172 |
| C | 128 | 148 |
| C++ | 55 | 60 |
| Cobol | 91 | 73 |
| Excel | 6 | 47 |
| Java | 53 | 60 |
| SQL | 12 | 39 |
| Perl | 27 | 60 |
| Visual Basic | 29 | 50 |

97 JANTZEN (2008). The waterfall process model is briefly discussed in Section 3.1.2.1.

98 BOEHM *et al.* (2000b).

99 BOEHM *et al.* (2000b); JONES (1996).

100 JONES (1996). In the given table, the values for "Assembler" and "Excel" refer to the values of "Assembly – Macro" and "Spreadsheet" of the original table given by JONES (1996) and BOEHM *et al.* (2000b). The value for SQL is taken from JONES (2008).

101 QUANTITATIVE SOFTWARE MANAGEMENT (2005). The presented values are taken from the column "Average."

### 2.2.2.3 Criticism on the Function Point Analysis

In contrast to its popularity and widespread use, the Function Point Analysis has been criticized since its initial publication. Based on a literature review, JEFFERY *et al.* identified *five* major criticisms related to the counting of Function Points[102]:

1) The level of abstraction is too high, turning a software system into a black box.[103] This is in line with BOEHM, for example, who postulates *"to work out as much detail as feasible."*[104] BOEHM argues that

   *"The more we explore, the better we understand the technical aspects [...]. The more pieces of software we estimate, the more we get the law of large numbers working for us [...]. The more we think through all the functions the software must perform, the less likely we are to miss [...] costs [...]."*[105]

2) The division of functionality into different User Function Types, i.e., Inputs, Outputs, Inquiries etc., may be technology dependent, and, in consequence, change with new technologies.[106] VERNER *et al.* argue that

   *"Modern interactive systems tend to blur these divisions. [...] If a new technology allows the user to do new things, or to do things differently, then it would seem to us that user function has changed with the technology. Its measurement must therefore change also."*[107]

3) The assessment of function types as simple, average, and complex appears to be inappropriate.[108] SYMONS argues that

   *"[The] classification of all system component types (input, outputs, etc.) as simple, average, or complex, has the merit of being straightforward, but seems to be rather oversimplified. A system component containing, say, over 100 data elements is given at most twice the points of a component with one data element."*[109]

102 JEFFERY *et al.* (1993).
103 VERNER, TATE (1987); VERNER *et al.* (1989).
104 BOEHM (1981), p. 316.
105 BOEHM (1981), p. 316.
106 VERNER *et al.* (1989).
107 VERNER *et al.* (1989), p. 377.
108 SYMONS (1988).
109 SYMONS (1988), p. 3.

4) According to ALBRECHT, the choice of weights *"was determined by debate and trial."*[110] Accordingly, there is doubt that these weights are valid in all circumstances.[111]

5) Generally, Function Point counts as a measure of size is *not* technology independent.[112]

In addition, there is a controversial discussion of the 14 processing complexity factors. While SYMONS asks for *more* factors or an open-ended approach respectively, JONES argues that 14 factors are *not* necessary to sufficiently reflect processing complexity.[113] During the validation of cost estimation models on the basis of empirical data, KEMERER found that *Unadjusted Function Points* have higher correlations with actual effort than *Adjusted Function Points*.[114] KEMERER concluded that the processing complexity factors contribute little to cost estimation processes.[115] Moreover, SYMONS argues that Function Points do not appear to be *summable* in the way one would expect.[116] Finally, the Function Point Analysis faced lots of criticism for not being applicable to all types of software.[117]

## 2.2.3 Variants of the Function Point Analysis

Due to the criticism, drawbacks, and specificity of the original Function Point Analysis, numerous variants of the FPA have been developed, for example, *Data Points, Feature Points, Object Points, SPR Function Points, 3D Function Points, Use Case Points*, as well as *COSMIC FFP*.[118] In 2008, JONES lists 22 variants of Function Point metrics.[119] A timeline-based overview of the history and evolution of Size Estimation methods is given at the end of this subsection (see Fig. 2.2; page 33).

### 2.2.3.1 Data Points

Inspired by the work of VERNER and TATE on size and effort estimation within the context of 4th generation development, HARRY SNEED developed the *Data Point*

[110] ALBRECHT (1979); ALBRECHT, GAFFNEY JR (1983); JEFFERY *et al.* (1993); SYMONS (1988).
[111] SYMONS (1988); VERNER *et al.* (1989).
[112] JEFFERY *et al.* (1993); SYMONS (1988); VERNER *et al.* (1989).
[113] JONES (1988); SYMONS (1988).
[114] KEMERER (1987).
[115] JEFFERY *et al.* (1993); KEMERER (1987).
[116] SYMONS (1988).
[117] CONTE *et al.* (1986); GALEA (1995); GRADY (1992); HETZEL (1993); JEFFERY *et al.* (1993); JONES (1988); JONES (1996); KAN (1993); VERNER *et al.* (1989); WHITMIRE (1992).
[118] BUNDSCHUH, DEKKERS (2008). *SPR (abbr.): Software Productivity Research. COSMIC (abbr.): Common Software Measurement International Consortium. FFP (abbr.): Full Function Point.*
[119] JONES (2008).

method as an approach to derive a size measure based on a software's data model.[120] Accordingly, this counting technique focuses on *data objects*, e.g., the user interface, as well as *data elements*, e.g., entities, attributes, keys, and relations. The size measure can be adjusted by eight *quality factors* and ten *project conditions*. The Data Point method has not gained international attention.[121]

### 2.2.3.2 Feature Points

In 1986, CAPERS JONES, the founder of *Software Productivity Research* (SPR), introduced the *Feature Point* method.[122] This variant of the IFPUG FPA extends the User Function Types by a new sixth type that accounts for *algorithms*. Additionally, the Feature Point method restricts itself to *two* general application characteristics. In the past, however, the method has been criticized for its vague definition of algorithms and its lack of correlations with other FPA variants.[123] Today, SPR comments on the Feature Point method as follows:

> *"Feature Points [...] are not in common use today. The approach was defined in response to concerns about the 'under representation' of certain types of functionality by the IFPUG method. The current view is that [...] this is not an appropriate way to produce estimates. The primary issue with the Feature Point approach [...] is that it destabilizes the emphasis on pure elementary process content. [...] There are almost no documented guidelines or standards regarding how to the theoretical construct of Feature Points to actual estimates in a consistent and repeatable way."*[124]

### 2.2.3.3 Object Points

In 1990, LUIZ LARANJEIRA primarily discussed the strengths and weaknesses of existing size estimation techniques with regard to *object-oriented* software development.[125] LARANJEIRA proposed a new method in order to improve estimation accuracy by taking the characteristics of object-oriented systems into account. One year later, BANKER *et al.* introduced the term *"Object Points."*[126] Another important pub-

120 SNEED (1990); SNEED (1996); VERNER, TATE (1988).
121 SNEED (1996).
122 JONES (1988); SOFTWARE PRODUCTIVITY RESEARCH (2009b).
123 BUNDSCHUH, DEKKERS (2008).
124 SOFTWARE PRODUCTIVITY RESEARCH (2009a).
125 LARANJEIRA (1990).
126 BANKER *et al.* (1991).

lication on object-oriented measurement was given by CHIDAMBER and KEMERER in 1994.[127]

Based on empirical data of 19 development projects, BANKER *et al.* found that *Object Points* correlated with effort just as well as *Function Points*.[128] Likewise, a successively conducted experiment-based study showed comparable estimation accuracy for Object Points and Function Points. The average time for creating Object Point estimates, however, was merely half the time it takes for creating Function Point estimates.[129]

It remains to be seen whether Object Points will gain widespread attention in the future, or whether FPA will keep its leading position.[130] Nevertheless, BOEHM *et al.* favor Object Points over Function Points with regard to the *COCOMO II Application Composition Model*. In this context, the Object Point counting focuses on the number of screens, reports, and 3rd generation language modules of a new software system, of which elements can be classified as *Simple*, *Medium*, or *Difficult*.[131]

#### 2.2.3.4 SPR Function Points

In 1985, SPR released the first commercial estimating tool that was based on Function Points.[132] This tool primarily supported Backfiring for converting LOC-based metrics into Function Points. The *SPR Function Point* method was developed by CAPERS JONES, who also developed the *Feature Point* method, in order to get a simplified variant of the *IFPUG Function Point* method.

The goal of that simplification was creating a method that allows producing estimates in less time compared to the traditional FPA.[133] Therefore, the complexity levels of FPA (i.e., simple, average, and complex) were eliminated.[134] Additionally, the 14 general application characteristics were reduced to three: *problem complexity,*

[127] BUNDSCHUH, DEKKERS (2008); CHIDAMBER, KEMERER (1994). Another object-oriented estimation approach that can be found in literature, e.g., in JANTZEN (2008) and BOEHM *et al.* (2000a), is *ObjectMetrix* maintained by TASSC (2009).
[128] BANKER *et al.* (1991); BOEHM *et al.* (2000b).
[129] BOEHM *et al.* (2000b); KAUFFMAN, KUMAR (1993).
[130] BUNDSCHUH, DEKKERS (2008).
[131] BOEHM *et al.* (2000b).
[132] JONES (2008).
[133] JONES (2008).
[134] BUNDSCHUH, DEKKERS (2008).

*code complexity,* and *data complexity.*[135] These characteristics are rated on a five-point integer-based scale, which also allows floating point ratings.

#### 2.2.3.5 3D Function Points

The *3D Function Point* (3D FP) method was developed at *Boing Computer Services* by SCOTT WHITMIRE in the early 1990s.[136] The method aimed at scientific and real-time software. Similar to Feature Points' consideration of algorithms, 3D FP takes the control flow deliberately into account.[137] Accordingly, the method estimates system size by analyzing three dimensions: *data, control,* and *process*. Today, the 3D Function Point method seems to be no longer deployed.[138]

#### 2.2.3.6 Use Case Points

The concept of *"Use Cases"* has been popularized by the *Unified Modeling Language* (UML).[139] Before UML has been developed in the 1990s, IVAR JACOBSON, who was one of the contributors of UML, developed the technique of specifying Use Cases.[140] According to JONES,

> *"A Use Case describes what happens to a software system when an actor (typically a user) sends a message or causes a software system to take some action."*[141]

The concept of *Use Case Points* (UCP) has been initially described by GUSTAV KARNER in 1993.[142] Goal of the UCP method is estimating effort. However, the method comprises a sizing technique that differs from traditional FPA.[143] The UCP estimation approach consists of four steps, that is, determining: 1) the *Unadjusted Use Case Points* (UUCP), 2) the *Technical Complexity Factor* (TCF), 3) the *Environmental Complexity Factor* (ECF), as well as 4) the *Productivity Factor* (PF).[144]

---

[135] JONES (2008) lists the given three characteristics. Probably incorrectly, BUNDSCHUH, DEKKERS (2008) only list two remaining characteristics. An introduction to SPR Feature Point counting can be found in JONES (2008), pp. 149-155.

[136] WHITMIRE (1992).

[137] BUNDSCHUH, DEKKERS (2008).

[138] JONES (2008).

[139] BALZERT (2005); FOWLER (2003); OBJECT MANAGEMENT GROUP (2004).

[140] JACOBSON *et al.* (1992).

[141] JONES (2008).

[142] KARNER (1993). A recent variant of the UCP method, especially for large software projects, is presented and discussed by FROHNHOFF (2008).

[143] BUNDSCHUH, DEKKERS (2008).

[144] JANTZEN (2008).

At first, a list of Use Cases and the involved actors is created. Each actor (humans or other systems) that interfaces with the software system is identified and categorized. Based on the *Actor Type*, categorized as simple, average, or complex, the actor receives a weighting that ranges from 1 to 3.[145] The sum of all weighted actors represents the *Unadjusted Actor Weight* (*$UAW_{total}$*).[146]

Similarly, each Use Case is identified and categorized. Based on the *Use Case Type*, the Use Case receives a weighting of 5, 10, or 15.[147] The sum of all weighted Use Cases represents the *Unadjusted Use Case Weight* (*$UUCW_{total}$*).

Given the weightings for actors and Use Case for each category, represented by $UAW_i$ and $UUCW_i$, and the numbers of actors and Use Cases per category, represented by $a_i$ and $u_i$, the total number of UUCP is calculated as follows[148]:

$$UUCP = UAW_{total} + UUCW_{total}$$

$$UAW_{total} = \sum_{i=1}^{3} UAW_i \times a_i \qquad \textbf{Eq. 2.2}$$

$$UUCW_{total} = \sum_{i=1}^{3} UUCW_i \times u_i$$

The Technical Complexity Factor is based on 13 complexity factors, e.g., *"Performance," "Reusability," "Portability,"* and *"Concurrency."* Their influences $I_i$ are rated on a scale from 0 ("no influence") to 5 ("strong influence").[149] Each complexity factor has a specific weight $w_i$ of either 0.5, 1.0, or 2.0 (see Tab. 2.3). "Portability" for example, is weighted by 2.0 while "Easy to install" is weighted with 0.5.[150] The TCF is calculated as follows:

$$TCF = 0.6 + 0.1\sum_{i=1}^{13} I_i \times w_i \qquad \textbf{Eq. 2.3}$$

145 SCHNEIDER, WINTERS (1998).
146 BUNDSCHUH, DEKKERS (2008); JANTZEN (2008).
147 SCHNEIDER, WINTERS (1998).
148 JANTZEN (2008).
149 JANTZEN (2008); SCHNEIDER, WINTERS (1998).
150 JANTZEN (2008); SCHNEIDER, WINTERS (1998).

**Tab. 2.3:** Technical complexity factors[151]

| Factor | Description | Weight |
|---|---|---|
| 1 | Distributed system | 2.0 |
| 2 | Performance | 1.0 |
| 3 | End user efficiency | 1.0 |
| 4 | Complex internal processing | 1.0 |
| 5 | Reusability | 1.0 |
| 6 | Easy to install | 0.5 |
| 7 | Easy to use | 0.5 |
| 8 | Portability | 2.0 |
| 9 | Easy to change | 1.0 |
| 10 | Concurrency | 1.0 |
| 11 | Special security features | 1.0 |
| 12 | Provides direct access for third parties | 1.0 |
| 13 | Special user training facilities are required | 1.0 |

Analogously, the Environmental Complexity Factor is based on 8 infrastructure factors, e.g., *"Familiarity with UML," "Analyst Capability,"* and *"Difficult Programming Language."* According to the rated influence, a factor can have a weight between -1.0 and +2.0 (see Tab. 2.4). While "Analyst Capability" has a positive effect on the ECP expressed by the negative weight -0.5, "Difficult Programming Language," for example, has a negative effect expressed by a positive weight of 1.0. The ECP is calculated by the following equation:

$$ECF = 1.4 - 0.03\sum_{i=1}^{8} I_i \times w_i \qquad \textbf{Eq. 2.4}$$

**Tab. 2.4:** Environmental complexity factors[152]

| Factor | Description | Weight |
|---|---|---|
| 1 | Familiarity with UML | 1.5 |
| 2 | Part-time workers | -1.0 |
| 3 | Analyst capability | 0.5 |
| 4 | Application experience | 0.5 |
| 5 | Object-oriented experience | 1.0 |
| 6 | Motivation | 1.0 |
| 7 | Difficult progr. languages | -1.0 |
| 8 | Stable requirements | 2.0 |

Given the results of UUCP, TCF, and ECF, it is possible to calculate the (adjusted) Use Case Points denoted by UCP: [153]

$$UCP = UUCP \times TCF \times ECF \qquad \textbf{Eq. 2.5}$$

For the sake of completeness, the final step of the UCP method aims at estimating the development effort by multiplying the overall UCP with the Productivity Factor.[154] Originally, the PF was set to 20 hours per UCP. SCHNEIDER and WINTERS

[151] BUNDSCHUH, DEKKERS (2008); JANTZEN (2008).
[152] BUNDSCHUH, DEKKERS (2008); JANTZEN (2008).
[153] BUNDSCHUH, DEKKERS (2008); JANTZEN (2008); SCHNEIDER, WINTERS (1998)
[154] The UCP method is criticized for the linear relationship of effort and size since linearity ignores the existence of organizational diseconomies of scale (JANTZEN (2008)).

suppose that a range from 20 to 28 h/UCP is realistic for most projects.[155] Finally, the overall effort is determined by the following equation:

$$Effort = UCP \times PF \qquad \textbf{Eq. 2.6}$$

It is obvious that the UCP method has a seamless transition from size estimation to effort estimation. The Unadjusted Use Case Points represent a pure size measure. Since the Technical Complexity Factor considers technical, requirement-specific factors only, it helps at adjusting the UUCP. The result is still a size measure. The Environmental Complexity Factor, however, introduces project- and developer-specific factors, which are only relevant when turning to effort estimation.

### 2.2.3.7 COSMIC FFP

*COSMIC FFP* is a composition of two acronyms, with one standing for the *"Common Software Measurement International Consortium"* and the other standing for the *"Full Function Point"* method developed by ALAIN ABRAN.[156] FFP was introduced in 1997 as an extension of the FPA for sizing real-time and technical software systems.[157] In 1998, the group around FFP merged with a working group of the *ISO/IEC Joint Technical Committee* in order to found COSMIC.[158] CHARLES SYMONS, who previously published the *Mark II Function Point* method, was member of this working group.[159] The name of the method has been lately changed from COSMIC FFP to COSMIC.[160]

In order to estimate the size of a new software system, the software is divided into unique functional processes. Each functional process consists of sub-processes, which refer to either data movements or data manipulations.[161] A data movement is based on elementary actions, i.e., *Read, Write, Entry,* and *Exit*. The actions Read and Write trivially refer to reading and writing data to persistent storages. The actions Entry and Exit describe data exchanges between the software and external entities, i.e., human users or other software systems.

---

155 SCHNEIDER, WINTERS (1998). The *Productivity Factor* is company-specific and must be gained by analyzing historical project data.

156 COSMIC (2007b).

157 COSMIC (2007b); ST-PIERRE *et al.* (1997).

158 COSMIC (2008). Working Group 12, ISO/IEC Joint Technical Committee 1, Sub-Committee 7.

159 COSMIC (2008); SYMONS (1988).

160 COSMIC (2007a). *ISO (abbr.): International Organization for Standardization, IEC (abbr.): International Electro-technical Commission*

161 COSMIC (2007b); JANTZEN (2008); VOGELEZANG (2006).

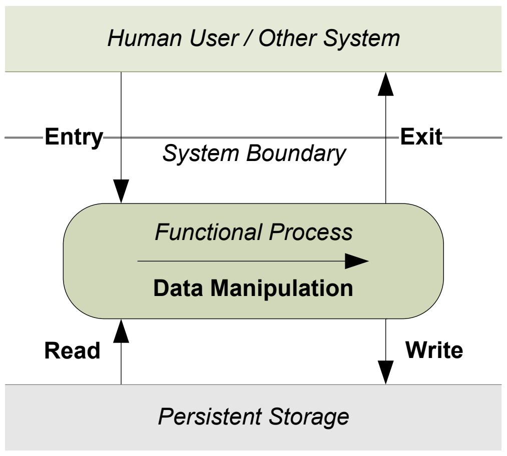


**Fig. 2.1:** Sizing model of COSMIC FFP[162]

The size of a new software system has formerly been measured in *COSMIC Functional Size Units* (CFSU), which have recently been renamed to *COSMIC Function Points* (CFP).[163] The size is determined by collecting all unique functional processes. For each process, the necessary data movements are counted. For example, a process consisting of one Entry and one Write has a size of two CFP.

It has to be pointed out that COSMIC FFP does not provide an approach to explicitly capture the number of required data manipulations. It is assumed that the average ratio of data movements and data manipulations is constant.[164] The assumed correlation of data movements and manipulations implicates that it is sufficient to determine data movements only in order to get an adequate size estimate. However, this assumption prevents utilizing COSMIC FFP for estimating the size of software systems that are dominated by data manipulations, for example, *algorithm-rich* applications.[165]

[162] COSMIC (2007b); JANTZEN (2008); VOGELEZANG (2006).
[163] COSMIC (2007a).
[164] VOGELEZANG (2006).
[165] COSMIC (2007b); COSMIC (2009).

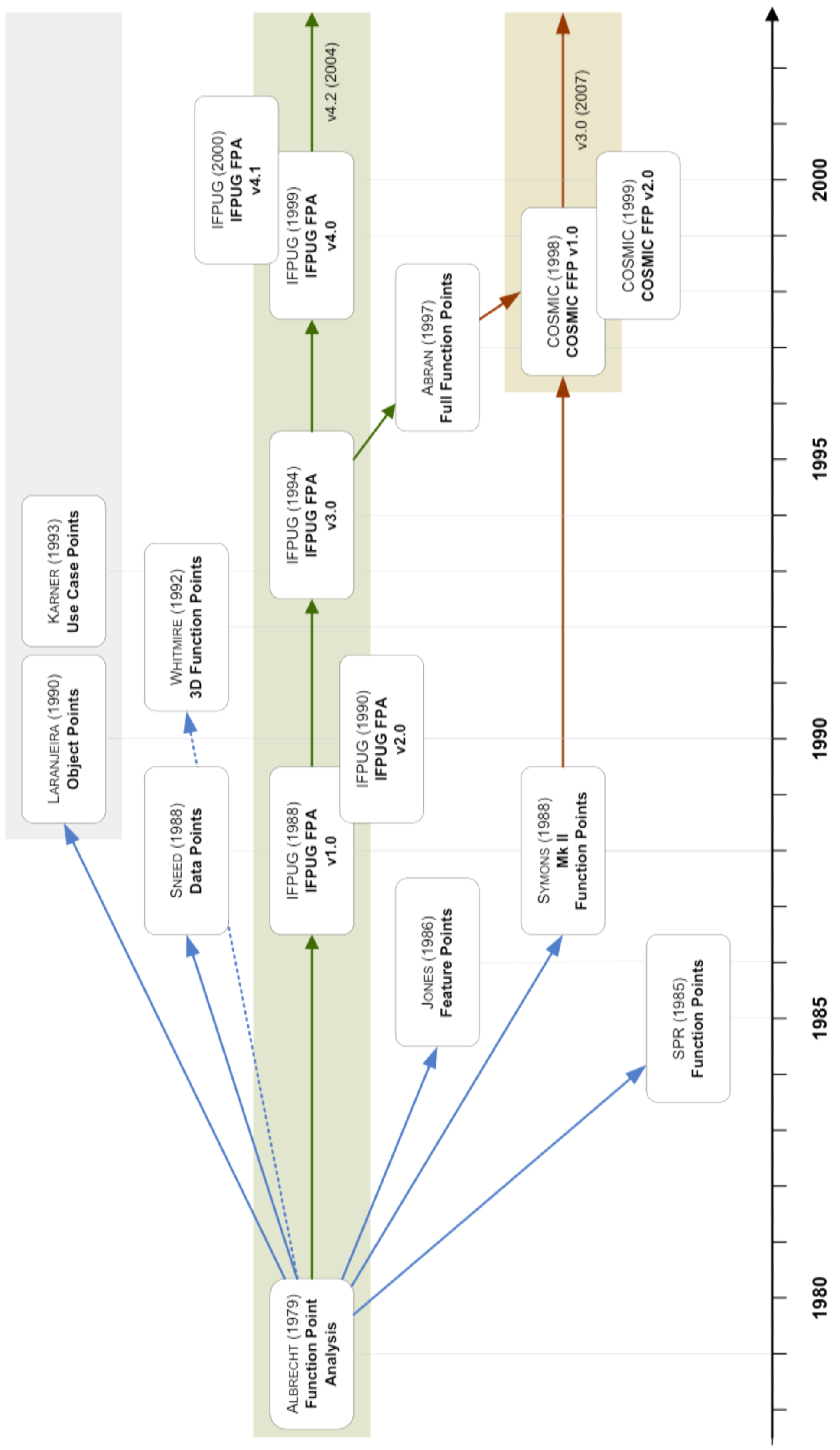

ALBRECHT (1979) Function Point Analysis
IFPUG (1988) IFPUG FPA v1.0
IFPUG (1990) IFPUG FPA v2.0
IFPUG (1994) IFPUG FPA v3.0
IFPUG (1999) IFPUG FPA v4.0
IFPUG (2000) IFPUG FPA v4.1
v4.2 (2004)
SNEED (1988) Data Points
WHITMIRE (1992) 3D Function Points
LARANJEIRA (1990) Object Points
KARNER (1993) Use Case Points
JONES (1986) Feature Points
ABRAN (1997) Full Function Points
SYMONS (1988) Mk II Function Points
COSMIC (1998) COSMIC FFP v1.0
COSMIC (1999) COSMIC FFP v2.0
v3.0 (2007)
SPR (1985) Function Points
1980
1985
1990
1995
2000

**Fig. 2.2:** History and evolution of size estimation methods

## 2.3 Software Effort Estimation

*Software Effort Estimation* or *Software Cost Estimation* respectively aims at forecasting the amount of effort that is necessary to build or maintain a software product. Similar to the Software Size Estimation, early and typically incomplete or vague information must be used to extrapolate total project effort. According to BOEHM *et al.*, the primary purpose of cost estimation techniques is *budgeting*.[166] Other important purposes are *tradeoff* and *risk analysis*, *project planning* and *control,* as well as *software improvement investment analysis.*[167]

The goodness of an estimation technique is based on its estimation *accuracy.*[168] The first significant research on software costs is an empirical study conducted by NELSON, which is based on 169 software development projects that were completed between 1964 and 1966.[169] The empirical data was primarily used to provide guidelines that help managers estimating development costs more accurately. To this day, the motivation to improve estimation accuracy has persisted. Software practitioners still voice concern over insufficient estimation results.[170] Correspondingly, BOEHM *et al.* argue:

> *"The fast changing nature of software development has made it very difficult to develop parametric models that yield high accuracy for software development in all domains. [...] [Therefore] one of the most important objectives of the software engineering community has been the development of useful models that constructively explain the development life-cycle and accurately predict the cost of developing a software product."*[171]

Recent research on software development productivity and cost-overruns reflect these concerns and their topicality.[172] Until now, 60-80% of all software development projects still exceed budgets and schedules with an average cost overrun of approx.

[166] BOEHM *et al.* (2000a). This publication is a shortened, slightly modified version of Chapter 2 of the dissertation by SUNITA CHULANI (1999).

[167] BOEHM *et al.* (2000a). *Tradeoff and risk analysis* are important to describe the cost and schedule sensitivities of software projects decisions. *Project planning and control* refers to cost and schedule breakdowns by components, subsystems etc. *Software improvement investment analysis* confronts costs with benefits of tools, reuse, and process maturity.

[168] BOEHM *et al.* (2000a).

[169] NELSON (1966).

[170] KEMERER (1987).

[171] BOEHM *et al.* (2000a), p. 178.

[172] GLASS (2005); JORGENSEN (2004); JORGENSEN, GRIMSTAD (2005); JORGENSEN, MOLOKKEN (2006); JORGENSEN, SHEPPERD (2007); MAXWELL, FORSELIUS (2000); MOLOKKEN, JORGENSEN (2003); PREMRAJ *et al.* (2005); PREMRAJ *et al.* (2004).

30% (see Section 3.1.3 for a detailed discussion of cost-effectiveness in software development).

Generally, the *uncertainty* of effort estimates decreases with project progress, because new and more precise information becomes available at later project stages. For that reason, the accuracy of effort estimates depends on both the applied technique as well as the point in time. The relationship of uncertainty and potential estimation accuracy is illustrated by the *Cone of Uncertainty* (see Fig. 2.3), which was initially described by BOEHM.[173] The term “Cone of Uncertainty,” however, was coined 15 years later by MCCONNELL.[174]

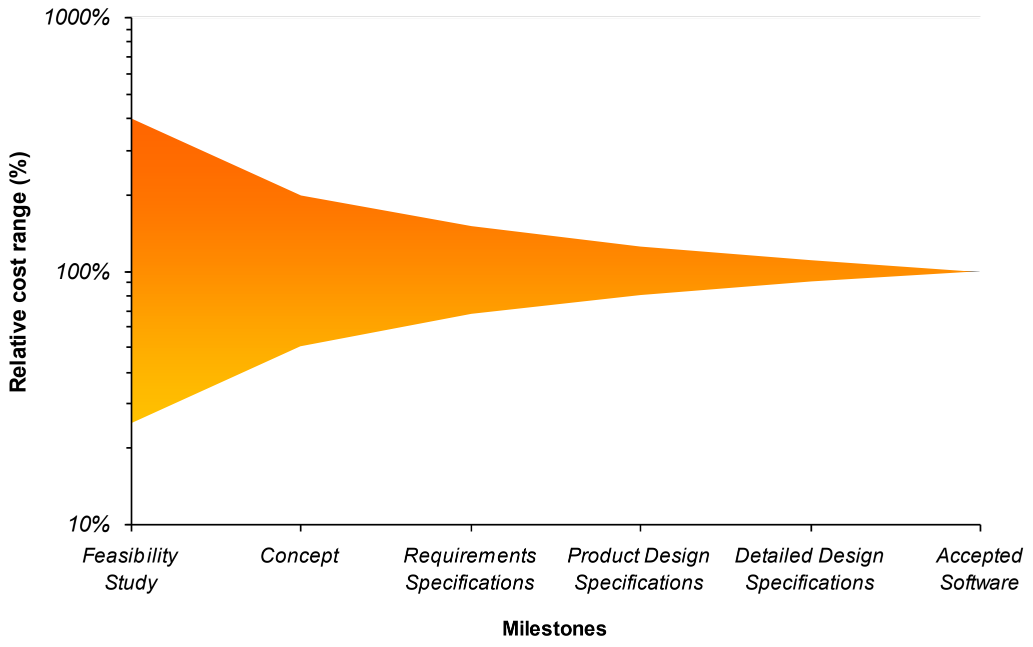


**Fig. 2.3:** Cone of uncertainty[175]

The ordinate of the corresponding diagram has a logarithmic scale. The diagram shows the likely deviation of the estimated costs depending on the project stage. At the beginning of a project (e.g., feasibility study) the uncertainty band is a factor of 16, since the probable cost range spreads from 25% to 400%. This implies that the actual costs are most likely between 25% and 400% of the initially estimated costs.[176]. In the subsequent concept phase, the uncertainty band reduces to 4, i.e., the actual costs are most likely between 50% and 200% of the estimated costs.[177] After

[173] BOEHM (1981).
[174] LITTLE (2006); MCCONNELL (1996).
[175] BOEHM (1981); MCCONNELL (1996).
[176] Similar cost ranges are described by LANDIS *et al.* (1990).
[177] LITTLE (2006).

the detailed specification, the relative cost range becomes manageable spreading from 80% to 125%. BOEHM argues that:

> *"If a software development cost estimate is within 20% of the 'ideal' cost estimate for the job, a good manager can turn it into a self-fulfilling prophecy."*[178]

In a survey on software cost estimation, BOEHM *et al.* present the most influential past and present estimation approaches.[179] In the following, this survey is used as an orientation for introducing the major estimation techniques. BOEHM *et al.* differentiate between *parametric*, *dynamics*-, and *regression*-based models as well as *expertise*-based and *learning*-oriented techniques.[180]

### 2.3.1 Model-based Techniques

This subsection focuses on PUTNAM'S *Software Life-cycle Model (SLIM)*[181] and BOEHM'S *Constructive Cost Model*[182] since these estimation techniques are well-documented and have been subject to empirical validation studies in the past.[183]

178 BOEHM (1981), p. 591.

179 BOEHM *et al.* (2000a). A similar, brief overview of cost estimation techniques is given by JANTZEN (2008).

180 BOEHM *et al.* (2000a). Regression-based techniques are not discussed in this work. Regression-based techniques are general, statistical approaches that allow building simple, formal models for effort estimation. Typically, such approaches use linear regression models that are based on the *Ordinary Least Squares* (OLS) method. In the context of software development effort estimation, the goal of regression models is to identify project-specific factors that explain the amount of development effort best. Another common application of regression-based techniques is the *calibration* of parametric cost models.
According to BOEHM *et al.* (2000a), OLS is only applicable, when 1) lots of empirical data is available, 2) no data is missing, 3) the sample does not contain outliers, 4) the input variables are uncorrelated, 5) the input variables are easy to interpret, and 6) the inputs are either all continuous or all discrete variables.

181 PUTNAM (1978).

182 BOEHM (1981); BOEHM *et al.* (2000b).

183 Model-based estimation models that are exclusively used in commercial or government contexts are not discussed in this work (e.g., *Checkpoint*, *PRICE-S, SEER-SEM, SELECT estimator*). Since these models are not completely released to the public, such techniques are neither well documented nor validated by research. Moreover, ESTIMACS developed by HOWARD A. RUBIN (1983) is also excluded from this discussion since a) the original publication of 1983 could not be obtained, b) all relevant publications that note ESTIMACS exclusively refer to that particular publication, c) there is no subsequent publication of RUBIN on ESTIMCAS, and d) ESTIMACS is not supported by commercial tools today. The basic idea of ESTIMACS is briefly described by BOEHM *et al.* (2000a) and CHULANI (1999).

#### 2.3.1.1 Software Life-cycle Model

In the 1970s, LAWRENCE PUTNAM developed the *Software Life-cycle Model* as one of the early cost estimation techniques.[184] SLIM is based on the findings of NORDEN who analyzed Research and Development (R&D) projects in the late 1960s. Based on his analyses, NORDEN developed a so-called *life-cycle pattern* in order to describe R&D projects quantitatively.[185] NORDEN found that the *life-cycle pattern* happens to follow a RAYLEIGH distribution, which is normally used in physics, e.g., for wind speed models.[186]

The mathematical foundation of SLIM is a consequence of the insufficient estimation approaches that were developed and discussed in the 1970s. Most generally, estimation models have been developed using multiple regression analysis which assumes *linear* relationships between input variables, e.g., work effort, and the output variable, e.g., delivered source lines of code (SLOC). Accordingly, MORIN commented on her study of existing cost estimating techniques published in 1973 as follows:

> *"I have failed to uncover an accurate and reliable method which allows programming managers to solve easily the problems inherent in predicting programming resource requirements. The methods I have reviewed contain several flaws [...] Researchers should apply nonlinear models of data interpretation [...] the use of nonlinear methods may not produce simple, quick-to-apply formulas for estimating resources, but estimation of computer resources is not a simple problem of linear cause-effect."*[187]

Agreeing with MORIN'S position on the preference of nonlinear estimation models, PUTNAM adopted the NORDEN/RAYLEIGH model to software development. In SLIM, the RAYLEIGH distribution is used to model the current manpower utilization over time.[188] This is achieved by the following differential equation:

$$\frac{dy}{dt} = 2Kate^{-at^2} \quad with \quad a = \frac{1}{2}t_d^2 \qquad \textbf{Eq. 2.7}$$

In this equation, *e* is EULER'S constant and *t* represents a point in time. The shape of the RAYLEIGH curve, and thus the size of the corresponding project, is determined by

184 PUTNAM (1978); PUTNAM, MYERS (1991).

185 NORDEN (1970).

186 Visually, the RAYLEIGH distribution appears similar to the normal distribution. However, the RAYLEIGH distribution is always right-skewed and, therefore, not symmetrical (cp. Fig. 2.4). Given two independent and normally distributed variables X and Y with zero means and identical variances, then $R = \sqrt{X^2 + Y^2}$ represents a RAYLEIGH distribution.

187 MORIN (1973).

188 PUTNAM (1978).

two further parameters: the total life-cycle effort $K$ as well as the point in time $t_d$ where the manpower utilization is at maximum.[189] The output of a software project is specified in SLOC. In order to estimate future project effort, historical data must be available to calculate a *productivity rate*. Moreover, the estimation approach needs a *technology constant* that describes the current state of technology.[190] With the help of this data, SLIM produces estimations of the amount and distribution of work effort. An exemplary curve is given in the following Fig. 2.4.

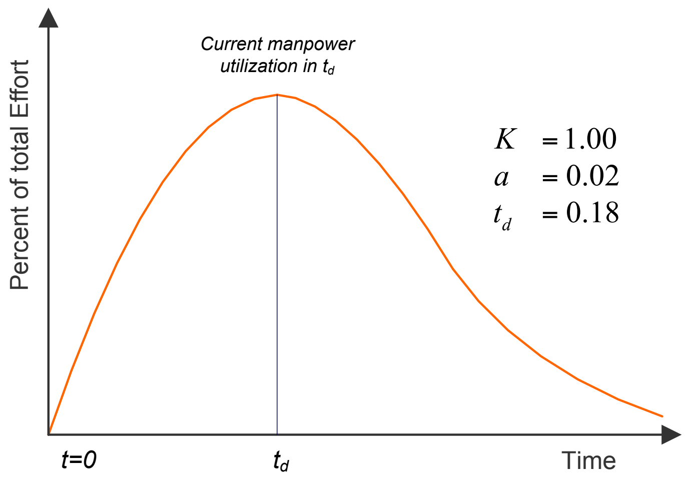


**Fig. 2.4:** The RAYLEIGH model[191]

It is obvious that SLIM uses a high degree of abstraction for modeling software development projects. As a consequence, this approach does not reflect different tasks and capabilities of involved developers. Instead, the organizational process is modeled as a black-box, which – as a whole – is supposed to follow a mathematical distribution. Thus, some of the assumptions do not always hold true. For example, *incremental and iterative development* methodologies cause *flat* staffing curves, which are contradictory to the RAYLEIGH curve.[192]

In order to comprehend the motivation and mathematical foundation of SLIM, it might be helpful to know that PUTNAM has graduated in *physics* and moved from nuclear engineering to management information systems during his career.[193] Interestingly, when publishing his work in the late 1970s, PUTNAM was optimistic about the accuracy of software cost estimation:

[189] PUTNAM (1978).
[190] PUTNAM (1978).
[191] BOEHM *et al.* (2000a); PUTNAM (1978).
[192] BOEHM *et al.* (2000a).
[193] PUTNAM (1978). The personal background of LAWRENCE H. PUTNAM is briefly summarized at the end of the referenced publication.

> *"Using the technique developed in the paper, adequate analysis for decisions can be made in an hour or two using only a few quick reference tables and a scientific pocket calculator."*[194]

Today, SLIM is still used and distributed in a set of commercial tools for planning, tracking, and benchmarking development projects.[195]

### 2.3.1.2 Constructive Cost Model

The *Constructive Cost Model* was primarily published in 1981 by BARRY BOEHM in his book *"Software Engineering Economics."*[196] In *Basic COCOMO*, the primary cost driver of a development project is defined by the number of thousands of *delivered source instructions* (KDSI). As a start, BOEHM introduces the fundamental equation of COCOMO for estimating the number of *man-month* (MM) based on the number of KDSI:

$$MM = 2.4(KDSI)^{1.05} \qquad \textbf{Eq. 2.8}$$

In addition to the estimation of effort, COCOMO provides another fundamental equation for estimating the *development schedule* (TDEV; *"time to develop"*):

$$TDEV = 2.5(MM)^{0.38} \qquad \textbf{Eq. 2.9}$$

The concept behind the estimation of TDEV is that the number of man-month cannot be shared by an arbitrary number of developers. This is because certain development tasks must be completed in a sequential order, and certain tasks might be indivisible. Moreover, adding developers to a project increases the communication overhead exponentially.[197]

A third equation is provided by basic COCOMO in order to estimate the average staffing of a project. FSP (*Full Time Equivalent Software Personnel*) represents the number of developers who are simultaneously on a project. BOEHM presents two alternative approaches to estimate the average FSP of a project phase.[198]

The first alternative is straightforward and utilizes arithmetic means. Given the project phase's percentage $p_e$ of total effort, its percentage of total schedule time $p_s$, and the project's total number of required *MM* as well as the corresponding *TDEV*, then *FSP* is estimated by the following equation:

---

194 PUTNAM (1978), p. 345.

195 BOEHM *et al.* (2000a); QUANTITATIVE SOFTWARE MANAGEMENT (2008).

196 BOEHM (1981).

197 BROOKS (1995b), p. 25 ("BROOKS' Law").

198 BOEHM (1981).

$$FSP = \frac{p_e MM}{p_s TDEV} \qquad \textbf{Eq. 2.10}$$

The second alternative for estimating the project staffing is mathematically more complex as it also makes use of the aforementioned RAYLEIGH distribution.[199] Similar to the ideas of PUTNAM, BOEHM proposes the RAYLEIGH curve as a good approximation of the labor distribution of software projects. The general equation reads as follows:

$$FSP = MM(\frac{t}{t_d^2})e^{-(t^2/2t_D^2)} \qquad \textbf{Eq. 2.11}$$

BOEHM recommends using the portion of the RAYLEIGH distribution between *0.3* $t_D$ and *1.7* $t_D$ as a sufficient approximation of the full development cycle.[200] In this case, the equation for estimating FSP of a certain point in time *t* is:

$$FSP = MM(\frac{0.15TDEV + 0.7t}{0.25(TDEV)^2})e^{-\frac{(0.15TDEV+0.7t)^2}{0.5(TDEV)^2}} \qquad \textbf{Eq. 2.12}$$

Fig. 2.5 visualizes the basic COCOMO estimates for so-called *organic-mode* development projects. While the graph of total effort is nearly linear, since the corresponding exponent for organic-mode projects is near one, the graphs of schedule and average staffing show a degressive growth.

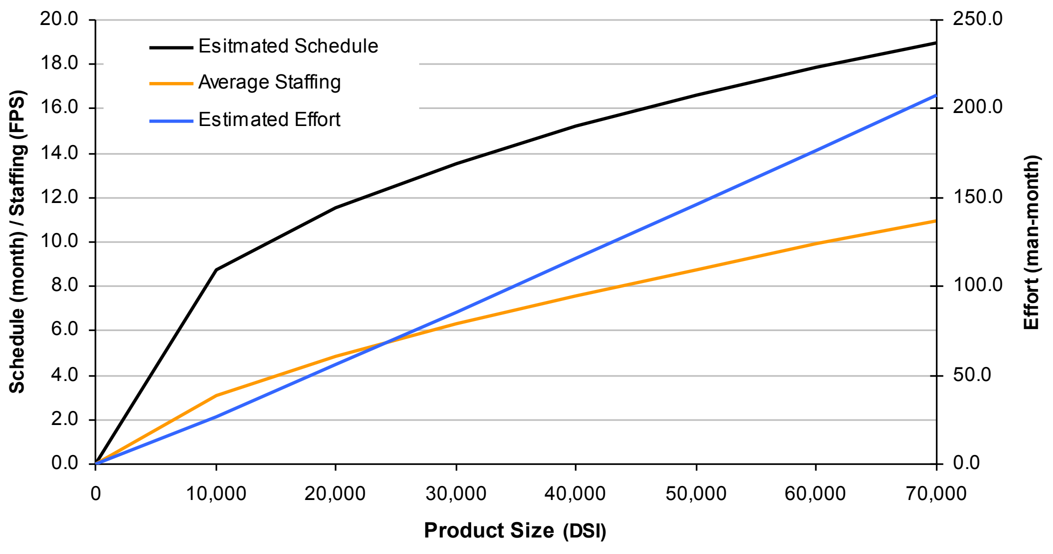


**Fig. 2.5:** Basic COCOMO estimates for organic mode projects[201]

[199] See Section 2.3.1.1.
[200] BOEHM (1981).
[201] A simplified version of this figure is given by BOEHM (1981).

### *Development Modes of Basic COCOMO*

So far, the presented equations used parameter values that are typical for *organic mode* development. During the development of COCOMO, BOEHM found that projects with the same size (measured in KDSI) can have different relationships of size and effort.[202] These different relationships helped identifying three typical development modes, which are briefly outlined in the following:

- The *organic mode* represents the most common development mode, which refers to in-house, familiar software development environments. Teams are relatively small and share extensive working experience. All team members have experience in working with related systems so that the system under development can easily be understood. Usually, each team member is able to contribute to the project while communication overhead is marginal. Moreover, even for larger organic-mode projects, communication overhead causes only a slight decrease of overall project productivity, and, therefore, organic-mode projects do not significantly suffer from organizational diseconomies of scale. BOEHM notes, however, that very few organic-mode projects exceeded 50 KDSI.[203]

- The *semidetached mode* is an intermediate stage that bridges organic and embedded mode development.[204] A project can be classified as a semidetached-mode project, if all project characteristics are on an intermediate level, or if the project is a mixture of organic and embedded mode characteristics. BOEHM gives some exemplary characteristics in order to explain the difference between semidetached- and organic-mode project in terms of *"working experience with related systems":* 1) All team members only have an intermediate experience level with related systems, 2) the teams consists of experienced and inexperienced developers, and/or 3) team members have experience with some aspects of the system, but not with all.[205] A semidetached mode project can reach 300 KDSI.

- *Embedded-mode* development is essentially characterized by tight constraints. These constraints are based on a complex system of hardware, software, regulations, and operational procedures, in

[202] BOEHM (1981).
[203] BOEHM (1981).
[204] BOEHM (1981).
[205] BOEHM (1981).

which the final software will be embedded.[206] The given constraints are supposed to be unchangeable during the whole project. Therefore, in contrast to the organic mode, embedded-mode projects do not have the option to negotiate software changes by modifying the requirements or the interface specifications during the project. Therefore, the costs of necessary internal changes and fixes are high. Besides, much effort is needed to assure that changes are made correctly. Likewise, much effort must be invested in assuring that the software meets its requirements. As a result, embedded-mode projects have a lower productivity and significantly face organizational diseconomies of scale.[207]

The different characteristics of *organic-mode, semidetached-mode,* and *embedded-mode* development projects are reflected by different coefficients and scale factors in the equations for estimating the overall effort (Eq. 2.8) and the development time (Eq. 2.9). The different equations are given in Tab. 2.5.

**Tab. 2.5:** Basic COCOMO effort and schedule equations[208]

| **Mode** | **Effort** | **Schedule** |
|---|---|---|
| Organic | $MM = 2.4(KDSI)^{1.05}$ | $TDEV = 2.5(MM)^{0.38}$ |
| Semidetached | $MM = 3.0(KDSI)^{1.12}$ | $TDEV = 2.5(MM)^{0.35}$ |
| Embedded | $MM = 3.6(KDSI)^{1.20}$ | $TDEV = 2.5(MM)^{0.32}$ |

The following diagram visualizes the different curves for each development mode:

[206] BOEHM (1981).
[207] BOEHM (1981).
[208] BOEHM (1981).

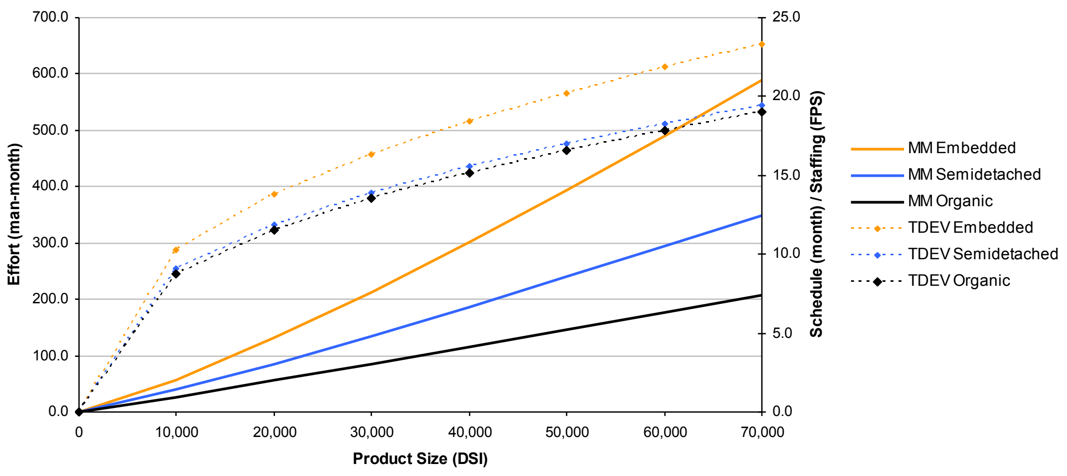


**Fig. 2.6:** Basic COCOMO estimates depending on development mode

### *Estimation Example*

In order to illustrate how the equations of COCOMO work together, BOEHM gives an example based on an organic, in-house development project of a chemical products company.[209] Given an initial study that roughly estimated a size of 32 KDSI, COCOMO provides the following estimates for the project:

$$\begin{aligned} &\text{Effort:} && MM = 2.4(32)^{1.05} = 91 && \text{[MM]} \\ &\text{Productivity:} && \frac{32{,}000}{91} = 352 && \text{[DSI/MM]} \\ &\text{Schedule:} && TDEV = 2.5(91)^{0.38} = 14 && \text{[month]} \\ &\text{Average staffing:} && \frac{91}{14} = 6.5 && \text{[MM/month; FSP]} \end{aligned} \qquad \textbf{Eq. 2.13}$$

### *Maintenance Effort*

In addition to the development estimators, *Basic COCOMO* offers another simple estimator for maintenance efforts. The estimation of maintenance effort requires the projected number of KDSI that will be added or modified during a year of maintenance. As a first step, this data is used to calculate the *Annual Change Traffic* (ACT)[210]:

$$ACT = \frac{KDSI_{added} + KDSI_{modified}}{KDSI_{development}} \qquad \textbf{Eq. 2.14}$$

[209] BOEHM (1981).
[210] BOEHM (1981).

Afterwards, the annual maintenance effort $MM_{AM}$ given in man-months is derived from the previously estimated number of development man-months $MM_D$.[211] The result provides the required annual maintenance staffing $FSP_{AM}$:

$$\begin{aligned} MM_{AM} &= ACT \cdot MM_D \\ FSP_{AM} &= MM_{AM} / 12 \end{aligned} \qquad \textbf{Eq. 2.15}$$

***Definitions and Assumptions of Basic COCOMO***

When using COCOMO as an instrument for development effort estimation, it is important to be familiar with the underlying definitions and assumptions.[212] BOEHM lists nine assumptions of which the most important ones are briefly summarized in the following:

- The primary cost driver is the number of delivered source instructions. The term *"delivered"* excludes, for example, source code that was written for testing, and, thus, is not part of the delivered product. The term *"source instruction"* refers to source lines of code without counting comment lines. It has to be pointed out that KDSI is a measure for the software size, which also has to be estimated.

- The development phase starts after the product design phase and ends with the software acceptance review (usually after integration and testing).

- A man-month consists of 152 hours of working time. A calendar month has 19 working days with 8 working hours per day.

- COCOMO assumes that the requirements specification is not significantly changed after the requirements phase.

The last assumption reflects that COCOMO assumes a waterfall development process, which is, however, not explicitly noted in the given list.[213]

211 BOEHM (1981).
212 BOEHM (1981).
213 BOEHM *et al.* (2000a); BOEHM *et al.* (2000b); JANTZEN (2008).

### 2.3.1.3 Intermediate and Detailed COCOMO

The limitations of *Basic COCOMO*, e.g., inability to accommodate iterative and incremental development as well as the missing incorporation of cost drivers beyond KDSI, have encouraged the development of more detailed estimation models. BOEHM comments on the benefits and limitations of *Basic COCOMO* as follows:

> *"Basic COCOMO is good for rough order of magnitude estimates of software costs, but its accuracy is necessarily limited because of its lack of factors to account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and other project attributes known to have a significant influence on costs."*[214]

In order to identify cost drivers other than KSDI, BOEHM examined former studies of WALSTON and FELIX[215], WEINWURM[216], as well as NESLON[217]. In total, these studies have identified over 100 different factors that are supposed to have an impact on development costs.[218] With the help of empirical project data, BOEHM eliminated factors from this list by statistically analyzing general *significance* and their *independence* from product size.

***Intermediate COCOMO***

Based on this analysis, 15 factors have been identified as relevant cost drivers, which have been incorporated in the *Intermediate COCOMO* model. During the estimation process, each cost driver is rated on a six-point scale, which ranges from *"very low"* to *"extra high."* A particular rated cost driver represents a so-called *effort multiplier* (EM). An overview of all 15 cost drivers and their effort multipliers is given in the following Tab. 2.6. The different values for each effort multiplier are the result of multiple regression analyses based on empirical project data. BOEHM gives brief descriptions of each cost driver and each rating in order to guide model users through the rating process.[219]

[214] BOEHM (1981), p. 58.
[215] WALSTON, FELIX (1977).
[216] WEINWURM (1970)
[217] NELSON (1966).
[218] BOEHM (1981).
[219] BOEHM (1981).

**Tab. 2.6:** Software development effort multipliers[220]

| Cost Drivers | | | Ratings | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | **Very low** | **Low** | **Nom.** | **High** | **Very High** | **Extra High** |
| *Product Attributes* | | | | | | | | |
| $EM_1$ | RELY | Required software reliability | 0.75 | 0.88 | 1.00 | 1.15 | 1.40 | - |
| $EM_2$ | DATA | Database size | - | 0.94 | 1.00 | 1.08 | 1.16 | - |
| $EM_3$ | CPLX | Product complexity | 0.70 | 0.85 | 1.00 | 1.15 | 1.30 | 1.65 |
| *Computer Attributes* | | | | | | | | |
| $EM_4$ | TIME | Execution time constraint | - | - | 1.00 | 1.11 | 1.30 | 1.66 |
| $EM_5$ | STOR | Main storage constraint | - | - | 1.00 | 1.06 | 1.21 | 1.56 |
| $EM_6$ | VIRT | Virtual machine constraint | - | 0.87 | 1.00 | 1.15 | 1.30 | - |
| $EM_7$ | TURN | Computer turnaround time | - | 0.87 | 1.00 | 1.07 | 1.15 | - |
| *Personnel Attributes* | | | | | | | | |
| $EM_8$ | ACAP | Analyst capability | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 | - |
| $EM_9$ | AEXP | Applications experience | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 | - |
| $EM_{10}$ | PCAP | Programmer capability | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 | - |
| $EM_{11}$ | VEXP | Virtual machine experience | 1.21 | 1.10 | 1.00 | 0.90 | - | - |
| $EM_{12}$ | LEXP | Progr. language experience | 1.14 | 1.07 | 1.00 | 0.95 | - | - |
| *Project Attributes* | | | | | | | | |
| $EM_{13}$ | MODP | Use of modern progr. practices | 1.24 | 1.10 | 1.00 | 0.91 | 0.82 | - |
| $EM_{14}$ | TOOL | Use of software tools | 1.24 | 1.10 | 1.00 | 0.91 | 0.83 | - |
| $EM_{15}$ | SCED | Required development schedule | 1.23 | 1.08 | 1.00 | 1.04 | 1.10 | - |

The nominal value for all effort multipliers is one. For the *"nominal"* case, the effort estimation of *Intermediate COCOMO* is identical to *Basic COCOMO*. If certain cost drivers are rated above or below the norm, they have an increasing or decreasing impact on effort. Therefore, the product of all effort multipliers is called *Effort Adjustment Factor* (EAF).[221] Given all rated cost drivers $EM_i$ as presented in Tab. 2.6, the EAF is calculated by the following equation:

$$EAF = \prod_{i=1}^{15} EM_i \qquad \textbf{Eq. 2.16}$$

In Intermediate COCOMO, the adjustment factor extends the equation for effort estimation as follows[222]:

$$MM = 2.4(KDSI)^{1.05} EAF \qquad \textbf{Eq. 2.17}$$

A simple min-max analysis of all possible values of the 15 effort multipliers shows a large interval for the EAF ranging from 0.097 to 72.379. Accordingly, since each

[220] BOEHM (1981).
[221] BOEHM (1981).
[222] In this equation, the given coefficient and scale factor refer to organic-mode development.

effort multiplier has a strong mathematical impact on the estimation equation, the rating of each cost driver must be done carefully.

For example, given two developer teams which are different in terms of analyst and programmer capability (ACAP, PCAP) – all other being equal, the estimated project effort for the less capable team doubles, while the estimated project effort for the more capable team decreases by 50%. Accordingly, different ratings of capability lead to an effort ratio of 4:1 between different teams.

***Detailed COCOMO***

The development of *Detailed COCOMO* is motivated by some limitations of *Intermediate COCOMO.* BOEHM argues that especially large projects suffer from *macro* estimations. For large projects, the estimated distribution of effort per development phase might be inaccurate. Moreover, the usability of *Intermediate COCOMO* is limited especially for projects with many components. In this case, working with *Intermediate COCOMO* can be cumbersome.[223]

The corresponding chapters on *Detailed COCOMO* introduce special *estimation forms* for large projects as well as detailed procedures that explain how to work with those forms. Four chapters focus on an in-depth explanation of the 15 effort multiplies and their rating definitions. However, the fundamental ideas of *Intermediate COCOMO* do not change. In view of that and, in particular, due to the presence of *COCOMO II*, *Detailed COCOMO* is not discussed in this work.

### 2.3.1.4 COCOMO II

The development of *COCOMO II* began in 1994 and was motivated by the limitations and drawbacks of the first *COCOMO* generation – often referred to as COCOMO'81. The first generation, for example, supported neither iterative, rapid development, nor object-oriented approaches.[224] *COCOMO II* was initially published in 1995.[225] The most extensive documentation*, "Software Cost Estimation with COCOMO II,"* was published by BOEHM *et al.* in 2000.[226]

223 BOEHM (1981).
224 BOEHM *et al.* (2000a).
225 BOEHM *et al.* (1995).
226 BOEHM *et al.* (2000b).

***COCOMO II Submodels***

COCOMO II provides three submodels, the *Application Composition Model*, the *Early Design Model*, and the *Post-Architecture Model*, each addressing different estimation issues.[227]

The *Early Design Model* has an explorative character. The model is used to make coarse-grained estimates at project start. Usually, much of a project is unknown and undefined at this point. The estimation is based on either Function Points or Lines of Code. Additionally, the estimation is influenced by five *scale factors* as well as seven *effort multipliers*.[228]

The Early Design Model is extended by the *Post-Architecture Model.* This model is used after the completion of top-level design, when detailed information about the project is available. In terms of structure and formulation, the Post-Architecture Model is close to the aforementioned *Intermediate COCOMO*. The estimation of the Post-Architecture Model is also based on Function Points or Lines of Code as well as five scale factors. In contrast to the Early Design Model, this model uses 17 effort multipliers.[229]

The *Application Composition Model* can be understood as an extension to the aforementioned models, which is especially used in ICASE[230] environments. These environments are characterized by the utilization of application frameworks, GUI builders, database management systems, domain-specific components or other reuse assets, as well as development tools for design, construction, and testing.[231] The effort estimation of the Application Composition Model is not based on SLOC or Function Points. Lines of Code become irrelevant when a significant amount of code is automatically generated. Additionally, Function Points cannot be applied since their level of abstraction does not match the level of generality and variability of application composition projects.[232] Therefore, size is expressed in *Object Points*.[233] BOEHM *et al.* suggest using the Application Composition Model in the first project stages since these development cycles are highly characterized by prototyping activities. Afterwards, the remaining project effort is estimated by the Early Design Model. Finally, when more accurate information is available, the effort estimation is adjusted by the Post-Architecture Model.

[227] BOEHM *et al.* (2000b).
[228] BOEHM *et al.* (2000a); BOEHM *et al.* (2000b).
[229] BOEHM *et al.* (2000a); BOEHM *et al.* (2000b).
[230] *ICASE (abbr.): Integrated Computer-Aided Software Engineering.*
[231] BOEHM *et al.* (2000b). *GUI (abbr.): Graphical User Interface.*
[232] BOEHM *et al.* (2000b).
[233] BANKER *et al.* (1991).

## *Effort Estimation Equation*

Both the Early Design and Post-Architecture models use the following equations for estimating effort and schedule. The amount of effort in person-month is estimated by:

$$PM_{NS} = A \times Size^{E} \times \prod_{i=1}^{n} EM_{i} \qquad \textbf{Eq. 2.18}$$

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_{j} \qquad \textbf{Eq. 2.19}$$

Generally, *Size* is determined by thousands of *Source Lines of Code* (KSLOC) or by *Unadjusted Function Points* (UFP). In the first equation, the subscript *NS* indicates a *nominal-schedule* estimate, which ignores the effects of schedule compression and stretch-out.[234] The parameter *A* represents a coefficient that is known from Eq. 2.8 of *Intermediate COCOMO*. The product of all effort multipliers has been introduced before as the *Effort Adjustment Factor* (cp. Eq. 2.16 and Eq. 2.17). However, the simple scale factors used in *Basic* and *Intermediate COCOMO* have been replaced by a more complex equation to calculate the exponent *E* (Eq. 2.19) which is based on a set of five special *Scale Factors* denoted by $SF_j$ (see Tab. 2.9). The equation is a replacement of the predefined development modes, i.e., organic, semidetached, and embedded mode, of COCOMO'81.[235] The parameter *n* in Eq. 2.18, as the upper bound of the summation of all effort multipliers $EM_i$, is 6 for the Early Design Model and 16 for the Post-Architecture Model.[236]

The values of the coefficient *A,* the constant *B,* the effort multipliers $EM_1, \ldots, EM_{16}$, as well as the scale factors $SF_1, \ldots, SF_5$ are the result of a model calibration based on empirical project data of 161 development projects.[237]

## *Effort Multipliers of COCOMO II*

When updating COCOMO'81 by COCOMO II, the set of effort multipliers was slightly modified. Two new product-specific factors were added, a few factors were

[234] BOEHM *et al.* (2000b). Nominal-schedule estimates implicate an exclusion of the cost driver for *Required Development Schedule* (SCED).

[235] BOEHM *et al.* (2000b).

[236] The Post-Architecture Model uses 16 (instead of 17) effort multipliers when making nominal-schedule estimates.

[237] BOEHM *et al.* (2000b). Experiences and strategies learned in the calibration of the COCOMO II Post-Architecture Model is given by CLARK *et al.* (1998). Further information on model calibration can be found in Chapter 4 in BOEHM *et al.* (2000b) which is based on CHULANI *et al.* (1999).

renamed.[238]. Moreover, all effort multiplier ratings have been recalibrated (see Tab. 2.7).

**Tab. 2.7:** Effort multipliers of COCOMO II[239]

| Cost Drivers | | | Ratings | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | Very low | Low | Nom. | High | Very High | Extra High |
| *Product Factors* | | | | | | | | |
| $EM_1$ | RELY | Required software reliability | 0.82 | 0.92 | 1.00 | 1.10 | 1.26 | - |
| $EM_2$ | DATA | Database size | - | 0.90 | 1.00 | 1.14 | 1.28 | - |
| $EM_3$ | CPLX | Product complexity | 0.73 | 0.87 | 1.00 | 1.17 | 1.34 | 1.74 |
| $EM_4$ | RUSE | Developed for reusability | - | 0.95 | 1.00 | 1.07 | 1.15 | 1.24 |
| $EM_5$ | DOCU | Documentation Match | 0.81 | 0.91 | 1.00 | 1.11 | 1.23 | - |
| *Platform Factors* | | | | | | | | |
| $EM_6$ | TIME | Execution time constraint | - | - | 1.00 | 1.11 | 1.29 | 1.63 |
| $EM_7$ | STOR | Main storage constraint | - | - | 1.00 | 1.05 | 1.17 | 1.46 |
| $EM_8$ | PVOL | Platform volatility | - | 0.87 | 1.00 | 1.15 | 1.30 | - |
| *Personnel Factors* | | | | | | | | |
| $EM_9$ | ACAP | Analyst capability | 1.42 | 1.19 | 1.00 | 0.85 | 0.71 | - |
| $EM_{10}$ | PCAP | Programmer capability | 1.34 | 1.15 | 1.00 | 0.88 | 0.76 | - |
| $EM_{11}$ | PCON | Personnel continuity | 1.29 | 1.12 | 1.00 | 0.90 | 0.81 | - |
| $EM_{12}$ | AEXP | Applications experience | 1.22 | 1.10 | 1.00 | 0.88 | 0.81 | - |
| $EM_{13}$ | PLEX | Platform experience | 1.19 | 1.09 | 1.00 | 0.91 | 0.85 | - |
| $EM_{14}$ | LTEX | Language and tool experience | 1.20 | 1.09 | 1.00 | 0.91 | 0.84 | - |
| *Project Factors* | | | | | | | | |
| $EM_{15}$ | TOOL | Use of software tools | 1.17 | 1.09 | 1.00 | 0.90 | 0.78 | - |
| $EM_{16}$ | SITE | Multisite development | 1.22 | 1.09 | 1.00 | 0.93 | 0.86 | 0.80 |
| $EM_{17}$ | SCED[240] | Required development schedule | 1.43 | 1.14 | 1.00 | 1.00 | 1.00 | - |

Based on the given *minima* and *maxima* in Tab. 2.7, it is possible to determine the productivity range for each cost driver. For example, the minimal impact on effort by the database size is 0.90 (*"low"* rating), while the maximal impact is 1.28 (*"very high"* rating). For that reason, the productivity range described by this cost driver is

[238] The two factors added reflect the degree of planned reusability (RUSE) and the required match of documentation to life-cycles (DOCU). The list of *computer attributes* was renamed to *Platform Factors*. The *Computer Turnaround Time* (TURN) was removed and the *Virtual Machine Constraint* (VIRT) was replaced by the *Platform Volatility* (PVOL). The list of *Personnel Factors* was extended by *Personnel Continuity* (PCON) and *Platform Experience* (PEXP). The factor *Programming Language Experience* (LEXP) was changed to *Language and Tool Experience* (LTEX). From the list of *Project Factors*, the *Use of Modern Programming Practices* (MODP) was removed. This list now contains the new factor *Multisite Development* (SITE)

[239] BOEHM *et al.* (2000b).

[240] The values for "nominal," "high," and "very high" are all 1.00 since schedule stretch-outs do not change effort.

1.28/0.90 = 1.42.[241] Fig. 2.7 gives an overview of the productivity ranges for each cost driver. In this diagram, the capability- and experience-oriented cost drivers are combined.[242]

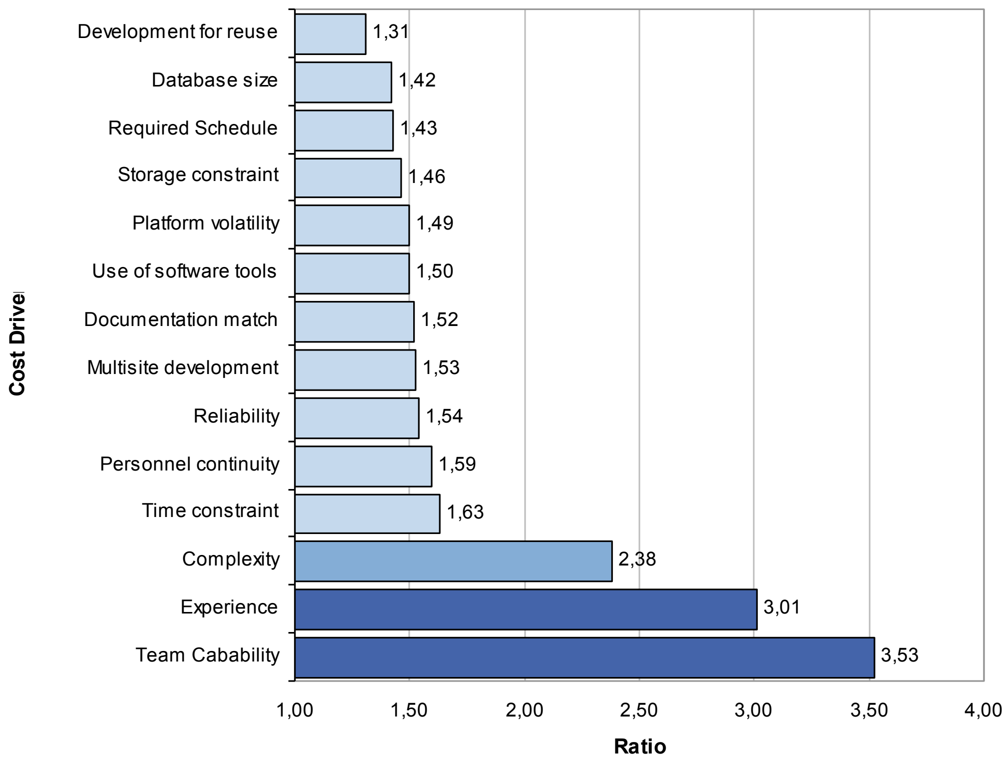


**Fig. 2.7:** Productivity ranges in COCOMO II[243]

### *Schedule Estimation Equation*

Similar to the effort estimation, the equation to estimate the calendar time, $TDEV_{NS}$, which is required to complete the project, has been slightly modified in COCO-MO II. The coefficient $C$ and the constant $D$ used in the following equations have also been obtained by the calibration of historical project data.[244] The other parameters are used as defined before (see Eq. 2.18 and Eq. 2.19).

$$TDEV_{NS} = C \times (PM_{NS})^{F} \qquad \textbf{Eq. 2.20}$$

241 A productivity range of 1.42 for one factor means that – all other being equal while the scale factor is 1.00 – the effort varies from 100% (lowest rating) to 142% (highest rating). In other words, the highest rating causes 42% more effort compared to the lowest rating.

242 ACAP and PCAP are combined as “Experience”. AEXP, PLEX, and LTEX are combined as “Team Capability”.

243 A comparable diagram is given on the front cover of BOEHM *et al.* (2000b).

244 BOEHM *et al.* (2000b).

$$\begin{aligned} F &= D + 0.2 \times 0.01 \times \sum_{j=1}^{5} SF_j \\ &= D + 0.2 \times (E - B) \end{aligned} \qquad \textbf{Eq. 2.21}$$

The calibrated values for *A, B, C* and *D* are given in the following Tab. 2.8:

**Tab. 2.8:** Calibrated parameter values of COCOMO II[245]

| A | B | C | D |
|---|---|---|---|
| 2.94 | 0.91 | 3.67 | 0.28 |

The scale factors $SF_j$ are rated on a six-point scale ranging from *"Very low"* to *"Extra High."* This type of rating scale has already been introduced in *Intermediate COCOMO* (cp. Tab. 2.6).[246] A high rating refers to a low value implicating positive scale effects. Therefore, scale factors that are rated as *"Extra high"* are expressed by zero.[247] The values for very low ratings range from 5.07 to 7.80 implicating significant diseconomies of scale. The scale factors used by COCOMO II are *Precedentedness* (PREC), *Development Flexibility* (FLEX), *Risk Resolution* (RESL), *Team Cohesion* (TEAM), and *Process Maturity* (PMAT).[248] Their values are given in the following Tab. 2.9.

---

245 BOEHM *et al.* (2000b). Also given in the *COCOMO II Manual* provided by USC-CSE (2008).

246 *Intermediate COCOMO* only uses a six-point rating scale to assess the cost drivers. The scale factor was determined by the development mode.

247 BOEHM *et al.* (2000b).

248 BOEHM *et al.* (2000b).

**Tab. 2.9:** Scale factor values SFj for COCOMO II models[249]

| Scale Factors | | Very Low | Low | Nominal | High | Very High | Extra High |
|---|---|---|---|---|---|---|---|
| PREC | | Thoroughly unprecedented | Largely un-precedented | | Generally familiar | Largely familiar | Thoroughly familiar |
| | | 6.20 | 4.96 | 3.72 | 2.48 | 1.24 | 0.00 |
| FLEX | | Rigorous | Occasional relaxation | Some relaxation | General conformity | Some conformity | General goals |
| | | 5.07 | 4.05 | 3.04 | 2.03 | 1.01 | 0.00 |
| RESL | | 20% | 40% | 60% | 75% | 90% | 100% |
| | | 7.07 | 5.65 | 4.24 | 2.83 | 1.41 | 0.00 |
| TEAM | | Very difficult interactions | Some difficult interactions | Basically cooperative interactions | Largely cooperative | Highly cooperative | Seamless interactions |
| | | 5.48 | 4.38 | 3.29 | 2.19 | 1.10 | 0.00 |
| PMAT[250] | | CMMI Level 1 | Upper CMMI Level 1 | CMMI Level 2 | CMMI Level 3 | CMMI Level 4 | CMMI Level 5 |
| | | 7.80 | 6.24 | 4.68 | 3.12 | 1.56 | 0.00 |

BOEHM *et al.* provide a simple example to illustrate the basic estimation of effort, schedule, and staff size with COCOMO II.[251] The approximated project size is 100 KSLOC. The project is assumed to have average characteristics. Thus, all effort multipliers are 1.0. In this example, the exponent *E* is manually set to 1.15 which represents a mixture of *"low"* and *"nominal"* ratings with regard to the five scale factors.[252] Applying the equations for estimating effort and calendar time as well as the calibrated parameter values (A=2.94, B=0.91, C=3.67, D=0.28), the estimated effort, project duration, and average amount of required developers are calculated as follows:

249 BOEHM *et al.* (2000b).

250 In BOEHM *et al.* (2000b), the process maturity is expressed in *SW-CMM* levels (Capability Maturity Model for Software). In the past year, however, *SW-CMM* has been stopped in favor of *CMMI* (Capability Maturity Model Integration).

251 BOEHM *et al.* (2000b). Also given in the *COCOMO II Manual* provided by USC-CSE (2008).

252 The values of the five scale factors rated as "nominal" range from 3.04 to 4.68 totaling to 18.97. The values of the five scale factors rated as "low" total to 25.28. An exponent of 1.15 implicates a total of 24.0 when B=0.91.

$$\text{Effort:}\quad PM_{NS} = 2.94 \times (100)^{1.15} \approx 586.61 \quad \text{[man-month]}$$

$$\begin{aligned}\text{Schedule:}\quad TDEV_{NS} &= 3.67 \times (586.61)^{0.28+0.2\times(1.15-0.91)} \\ &= 3.67 \times 586.61^{0.328} \\ &\approx 29.7 \quad \text{[month]}\end{aligned} \qquad \textbf{Eq. 2.22}$$

$$\text{Average Staff:}\quad \frac{\mathrm{PM_{NS}}}{\mathrm{TDEV_{NS}}} = \frac{586.61}{29.7} \approx 20 \quad \text{[developers]}$$

Finally, in order to illustrate the impact of the experience- and capability-oriented factors, another effort estimation example is briefly discussed. The approximated project size is again 100 KSLOC. The exponent *E*, i.e., the sum of all five scale factors, is assumed to be 1.05, which implies little diseconomies of scale. The effort is estimated for two different teams. One team gets the highest possible ratings for experience and capability, while the other team gets the lowest ratings. Accordingly, the effort estimations are influenced by all effort multipliers listed as *Personnel Factors* except for *Personnel Continuity*. The efforts for both teams are estimated as follows:

$$\begin{aligned}\text{Team 1:}\quad PM_{NS} &= 2.94 \times (100)^{1.05} \times 0.71 \times 0.76 \times 0.81 \times 0.85 \times 0.84 \\ &\approx 2.94 \times 125.9 \times 0.312 \\ &\approx 115.5 \quad \text{[man-month]} \\ \text{Team 2:}\quad PM_{NS} &= 2.94 \times (100)^{1.05} \times 1.42 \times 1.34 \times 1.22 \times 1.19 \times 1.20 \\ &\approx 2.94 \times 125.9 \times 3.315 \\ &\approx 1{,}227.0 \quad \text{[man-month]}\end{aligned} \qquad \textbf{Eq. 2.23}$$

The effort estimates demonstrate the strong impact of developer experience and capability in COCOMO II. The experienced team needs approx. 115.5 person-months to complete the project. In contrast, the inexperienced team needs 1,227 person-months which is a tenfold increase of estimated effort.

### 2.3.2 Expertise-based Techniques

According to BOEHM *et al.*, expertise-based estimation methods are useful if quantified, empirical data is not at hand.[253] Estimates are produced on the basis of *experience* of completed and comparable projects. JORGENSEN and BOEHM identify the *quantification step* as the essential difference between formal model-based and ex-

[253] BOEHM *et al.* (2000a).

pert-judgment-based effort estimation.[254] The quantification step refers to the *final* step of the estimation process that transforms the input into the quantitative effort measure.[255]

Following JORGENSEN, experts in this context might be *"all individuals with competence in estimating software,"* not only software development professionals but also students with sufficient experience in effort estimation.[256] However, BOEHM *et al.* point out that:

> *"The obvious drawback of this method is that an estimate is only as good as the expert's opinion [...]. Years of experience do not necessarily translate into high levels of competency. Moreover, even the most highly competent of individuals will sometimes simply guess wrong."*[257]

The characterization of expertise-base techniques ranges from *"unaided intuition ('gut feeling') to expert judgment supported by historical data, process guidelines, and checklists ('structured estimation')."*[258] Correspondingly, JORGENSEN and BOEHM argue that *"[...] there are many other ways of improving judgment-based processes than selecting the best expert."*[259] Nonetheless, SIMON explains the necessity of expertise-based analyses and the need to make decision based upon them:

> *"Intuition and judgment – at least good judgment – are simply ana1yses frozen into habit and into the capacity for rapid response through recognition. Every manager needs to be able to analyze problems systematically (and with the aid of the modem arsenal of ana1ytical tools provided by management science and operations research). Every manager needs also to be able to respond to situation rapid1y, a skill that requires the cu1tivation of intuition and judgment over many years of experience and training. The effective manager does not have the luxury of choosing between 'ana1ytic' and 'intuitive' approaches to problems."*[260]

In the following, the Delphi technique, the Work Breakdown Structure, and the Three-Point estimation are described. These techniques have in common that they

[254] JORGENSEN, BOEHM (2009).
[255] JORGENSEN (2007).
[256] JORGENSEN (2007), p. 450.
[257] BOEHM *et al.* (2000a), p. 192.
[258] JORGENSEN (2004), p. 37.
[259] JORGENSEN, BOEHM (2009), p. 3.
[260] SIMON (1987), p. 63.

are based on expertise on the on hand. However, these techniques also try to mitigate the problem of far out estimates and mistaken guesses on the other hand.

#### 2.3.2.1 Delphi Technique

The *Delphi Technique* originally comes from a military context. In the 1950s, the technique was developed by HELMER and DALKEY at the *RAND Corporation*[261]*,* an organization working closely with the U.S. armed forces.[262] Basically, the Delphi Technique was developed as a method of making predictions about future events. For that reason, the technique was named after the famous *oracle* located near the ancient, Greek town of *"Delphi."*[263] An in-depth introduction to the Delphi technique is given by LINSTONE and TUROFF.[264] In 2002, they list over 1,000 publications that have been addressing the subject since its development, which gives evidence for Delphi's general topicality and widespread use.[265]

A variant of Delphi, called *Wideband Delphi,* aims at guiding a group of individuals to a consensus of opinion.[266] Typically, the estimation process is conducted and supervised by a moderator.[267] In a first round, each participant is separately and confidentially asked for giving his or her opinion on an issue. Next, the individual responses are collected, anonymized, tabulated, and handed out to all participants.[268]

In a second round, all participants are again asked for giving their opinions. At this time, each participant knows the other participants' responses given in the first round. This knowledge usually influences the participants, reducing the variance of responses, and, thus, moves the opinions towards a reasonable middle ground.[269] This effect is related to the *Wisdom of the Crowds,* which became popular by the *"Ask the Audience"* lifeline in the TV show *"Who Wants to Be a Millionaire."*[270]

In contrast to the traditional Delphi technique, group discussions are permitted and encouraged by Wideband Delphi.[271] Based on this procedure, the group is supposed to reach a consensus on the issue of interest. In the context of software development,

[261] RAND is the acronym for *"Research and Development."*
[262] CUSTER *et al.* (1999); DALKEY, HELMER (1963).
[263] BOEHM *et al.* (2000a).
[264] LINSTONE, TUROFF (1975).
[265] LINSTONE, TUROFF (2002).
[266] BOEHM *et al.* (2000a); BOEHM (1981).
[267] MOLOKKEN-OSTVOLD, JORGENSEN (2004).
[268] BOEHM *et al.* (2000a).
[269] BOEHM *et al.* (2000a).
[270] SUROWIECKI (2004).
[271] BOEHM (1981); MOLOKKEN (2004).

this round-based procedure is used to narrow the estimated range of software size or development effort.

Whether the Delphi or Wideband Delphi techniques are used in software development practice is not well-known. Based on a review of surveys on software effort estimation[272], MOLOKKEN notices that

> *"To the best of our knowledge none of [...] the Delphi or the Wideband Delphi techniques has been subject to extensive empirical research in a software engineering context during the last 25 years."*[273]

Likewise, BOEHM *et al.* only list a few situations in which they have used the Wideband technique.[274]

### 2.3.2.2 Work Breakdown Structure

The *Work Breakdown Structure* (WBS) has its roots in the domain of project management. According to KERZNER:

> *"The Work Breakdown Structure acts as a vehicle for breaking the work down into smaller elements, thus providing a greater probability that every major and minor activity will be accounted for."*[275]

Generally, the goal of the WBS is organizing project elements into a hierarchy.[276] The most common variant of the WBS uses six hierarchy levels, e.g., *Task*, *Subtask*, and *Work Package*, which can be visualized – at least for simple projects – as a tree diagram. In order to plan and control costs, the expected effort is attributed to the bottom elements of the WBS hierarchy. Elements of a superordinated hierarchy level inherit the totalized efforts of the subordinated elements.[277] Since the recursive breakdown of a project into elementary work packages as well as the specification of components require both knowledge and experience, the WBS is categorized as an expertise-based technique.[278]

In the context of software development, a WBS usually consists of two hierarchies. One hierarchy describes the solution structure of the software, the other hierarchy

[272] MOLOKKEN, JORGENSEN (2003).
[273] MOLOKKEN (2004), p. 119.
[274] BOEHM *et al.* (2000a).
[275] KERZNER (1984), p. 515.
[276] BOEHM *et al.* (2000a).
[277] KERZNER (1984).
[278] BOEHM *et al.* (2000a).

focuses on the activities required for building the software.[279] The activity hierarchy allows fine-grained effort estimation as well as fine-grained monitoring and evaluating actual efforts. An exemplary software WBS contrasting the product and activity hierarchies is given in Fig. 2.8.

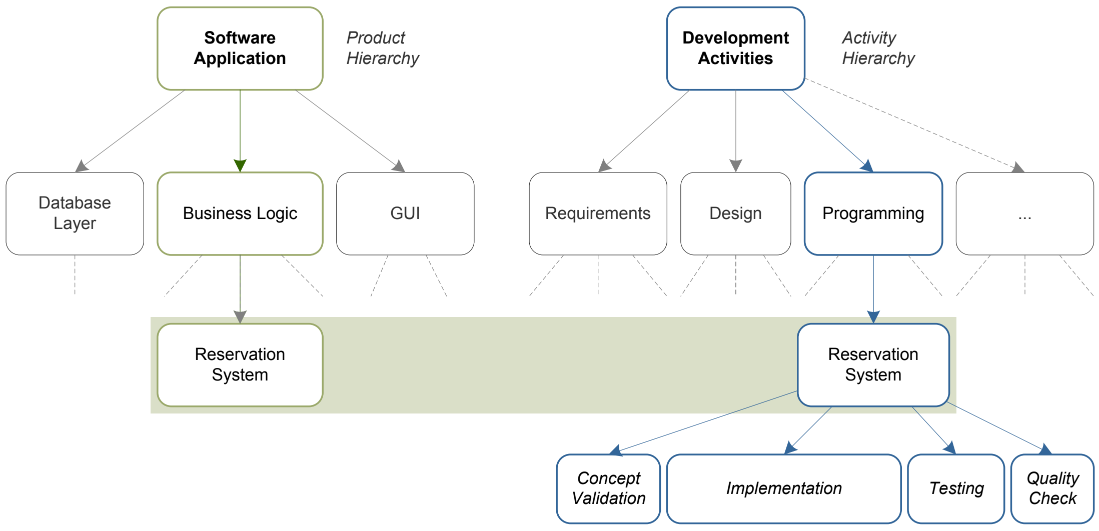


**Fig. 2.8:** Exemplary Work Breakdown Structure

### 2.3.2.3 Three-Point Estimate

The *Three-Point* or *Three-Time Estimate*, respectively, was developed as part of the *Program Evaluation and Review Technique* (PERT).[280] In the late 1950s, PERT was designed for modeling and managing large, complex projects. Among other features, the technique allows modeling so-called *PERT Networks* that are based on a list of activities and their predecessors. The PERT Network method is related to scheduling techniques like the *Arrow Diagram Method*, the *Precedence Diagram Method*, and the *Critical Path Method* (CPM).[281]

Essentially, the Three-Point Estimate presumes that single-point estimates are biased by optimism. In this context, McConnell argues that:

> *"Developers present estimates that are optimistic. Executives like the optimistic estimates because they imply that desirable business targets are achievable. Managers like the estimates because they imply that they can support upper management's objectives. And so the software project is off and running with no one ever taking a*

[279] Boehm *et al.* (2000a).
[280] McConnell (2006).
[281] Kerzner (1984).

*critical look at whether the estimates were well founded in the first place."*[282]

In order to mitigate this type of optimism bias, the expected effort $EE_i$ of an activity $A_i$ is calculated on the basis of three estimates: 1) The *Most Likely Case* ($M_i$) assumes that everything proceeds normally, 2) the *Optimistic Case* ($O_i$) or best case, respectively, is based on the theoretically *minimal* time required for task completion, 3) the *Pessimistic Case* ($P_i$) or worst case, respectively, assumes that everything goes wrong. The mitigation of optimism bias is based on confronting the *Most Likely* and *Optimistic Case*, which are usually close to each other, with the typically remote *Pessimistic Case* (see Fig. 2.9):

*"When developers are asked to provide single-point estimates, they often unconsciously present Best Case estimates."*[283]

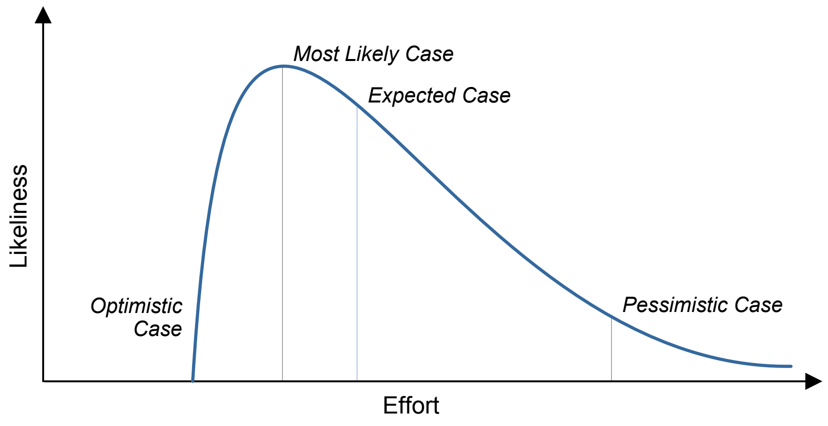


**Fig. 2.9:** Distribution of Optimistic, Most Likely, and Pessimistic Case

Given the estimates for Optimistic, Most Likely, and Pessimistic Case, the following equation determines the expected effort per activity:[284]

$$EE_i = \frac{O_i + 4M_i + P_i}{6} \qquad \textbf{Eq. 2.24}$$

The standard deviation (SD) of the total expected effort ($EE$) required for completing all $n$ activities can be determined by the next (simplified) equation:[285]

282 McConnell (2006), p. 47.
283 McConnell (2006), p. 119.
284 McConnell (2006).
285 This equation is only valid for a number of activities less than 10. McConnell (2006) also provides a more complex approach for larger numbers.

$$SD = \frac{\sum_{i=1}^{n} O_i - P_i}{6} \qquad \textbf{Eq. 2.25}$$

The standard deviation can be utilized to get a particular likelihood, which is 50% for the total effort *EE*. The sum of *EE+SD*, for example, has a likelihood of 84%, while the sum of *EE+2SD* reaches 98%.[286]

## 2.3.3 Learning-oriented Techniques

### 2.3.3.1 Case Studies

The purpose of *Case Studies* is *learning* useful lessons by generalizing and extrapolating from specific examples, i.e., former development projects.[287] Estimating development effort by examining cases is the oldest manual estimation technique:[288]

> *"Case studies represent an inductive process, whereby estimators and planners try to learn useful general lessons and estimation heuristics by extrapolation from specific examples. They examine [cases] describing the environmental conditions and constraints [...] during the development of previous software projects, the technical and managerial decisions that were made, and the final successes or failures that resulted. [...] Ideally they look for cases describing projects similar to the project for which they will be attempting to develop estimates, applying the rule of analogy that says similar projects are likely to be subject to similar costs and schedules."*[289]

The motivation and central idea of estimating development effort by case studies is similar to the Case Study research method. During a Case Study, the observer gains detailed insight by asking *"how"* and *"why"* questions.[290] A Case Study can give answers, for example, *why* specific process models, frameworks, and development tools are chosen, *how* teams are selected, *why* certain technical or managerial decision have been made, and *how* problems are approached. According to YIN:

> *"The distinctive need for case studies arises out of the desire to understand complex social phenomena. In brief, the case study allows an investigation to retain the holistic and meaningful charac-*

[286] MCCONNELL (2006).
[287] BOEHM *et al.* (2000a).
[288] Boehm *et al.* (2000a).
[289] Boehm *et al.* (2000a), p. 194.
[290] YIN (2003).

> *teristics of real-life events – such as individual life cycles [as well as] organizational and managerial processes [...].”*[291]

Such investigations aim at discovering links between cause and effect that are also valid in other contexts.[292] Preferably, the project to be estimated is similar to the observed project, in a way that *analogies* can be successfully applied. BOEHM *et al.* favor *“homegrown”* cases, which have been conducted in-house, since new projects will be most probably run in similar environments and under comparable conditions.[293] CHULANI briefly outlines the use of *Case-Based Reasoning* (CBR) as a formalized learning-oriented estimation technique:[294]

> *“Case-based reasoning is an enhanced form of estimation by analogy. A database of completed projects is referenced to relate the actual costs to an estimate of the cost of a similar new project. Thus, a sophisticated algorithm needs to exist which compares completed projects to the project that needs to be estimated.”*[295]

#### 2.3.3.2 Neural Networks

A *Neural Network* (NN) is an artificial, computational model that simulates biological neural networks. Basically, a Neural Network consists of linked, artificial *neurons,* which are typically grouped to input, hidden, and output layers.[296] Depending on the network structure, different network types can be identified. In contrast to *recurrent* networks, *Feed-Forward Networks* represent a directed acyclic graph. Information is forwarded in one direction only, consecutively processed by the input, hidden, and output neurons (see Fig. 2.10).[297]

By reconfiguring its neurons, a Neural Network is able to *learn*. Since inputs and outputs are *metric*[298], it is possible to quantify the prediction error, for example, by calculating the mean-squared deviation. The NN can be trained by propagating the

[291] YIN (2003), p. 14
[292] BOEHM *et al.* (2000a).
[293] BOEHM *et al.* (2000a).
[294] CHULANI (1999); SHEPPERD, SCHOFIELD (1997).
[295] CHULANI (1999), pp. 30-31.
[296] A neural network does not necessarily consist of three layers. Single-layer networks (*Perceptrons*), for example, are restricted to one input layer. Conversely, neural networks can have multiple hidden layers for information processing. However, with regard to the *Universal Approximation Theorem*, most problems require only one hidden layer (e.g., FUNAHASHI (1989)).
[297] HAYKIN (1998).
[298] Nominal inputs, for example, can be represented by a set of binary variables.

error within the network in a way that neurons can conform themselves to the new situation. A common and well-known training approach is *Backpropagation*.[299]

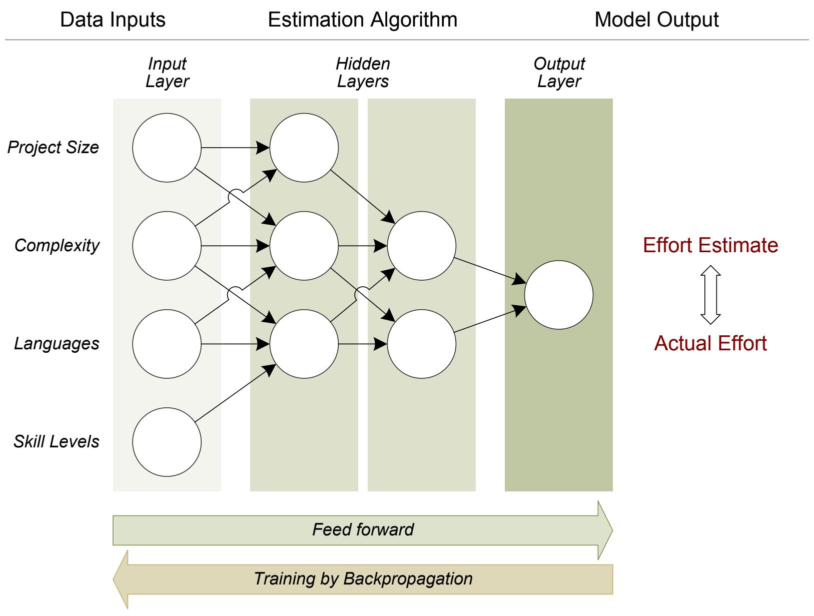


**Fig. 2.10:** Exemplary Feed-Forward Backpropagation Neural Network[300]

Software effort estimation can be realized by feeding an NN with historical project data. As illustrated in Fig. 2.10, data like project size, problem complexity, or staff skill can be used as model input. The error is determined by comparing the model output with the actual project effort. The NN learns and improves its prediction accuracy by iterating over the training data set until a predefined, minimal delta is reached. This delta prevents the network form overtraining.[301]

In a study on the prediction of program faults, KHOSHGOFTAAR *et al.* identified accuracy deficiencies of traditional multiple-regression methods compared to Neural Networks.[302] They conclude that:

> *"[...] neural network methods produce predictive models having better predictive quality than models produced with traditional multivariate regression methods. [...] Neural network models [...] are particularly suited for modeling of software complexity data, and offer an effective alternative to regression techniques that are often weakened in this environment by assumptions that are not met."*[303]

---

299 RUMELHART *et al.* (1986).
300 Cp. BOEHM *et al.* (2000a); CHULANI (1999).
301 BOEHM *et al.* (2000a).
302 KHOSHGOFTAAR *et al.* (1995)
303 KHOSHGOFTAAR *et al.* (1995), p. 153.

In a comparative study on common cost estimation techniques, BRIAND *et al.* provide a noteworthy comparison of previous research:[304] MUKHOPADYAY *et al.* found that their analogy-based model using CBR outperformed COCOMO.[305] Based on empirical data of 299 projects, FINNIE *et al.* compared CBR with different regression models. Once again, CBR showed a better accuracy than regression models. However, Neural Networks outperformed CBR.[306]

## 2.3.4 Dynamics-based Techniques

In the mid-1950s, JAY FORRESTER started the primary research on *System Dynamics*. The fundamental publications were released in the 1960s.[307] *System Dynamics* is a simulation methodology that models complex problems as systems of *Feedback Loops* as well as so-called *Stocks* and *Flows* (see Fig. 2.11). FORRESTER developed *System Dynamics* as an instrument to simulate, understand, and improve complex organizations, especially, from the managerial perspective.[308] Simulation runs are realized by mathematical models defined by sets of first order differential equations.[309]

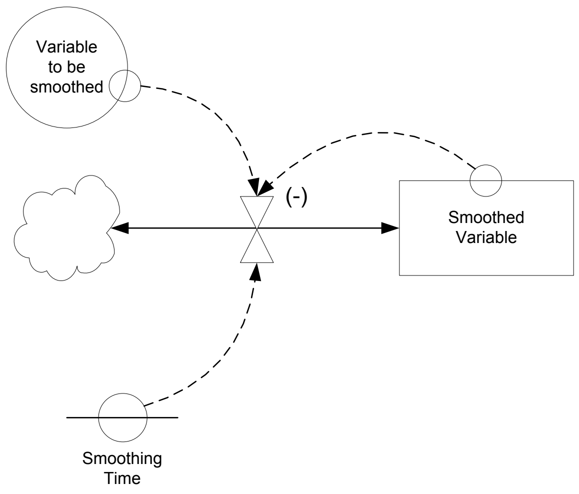


**Fig. 2.11:** Exemplary System Dynamics model notation[310]

In contrast to the effort estimation approaches presented before, dynamics-based techniques take into account that certain factors are *not static*, but *change dynamically* during a project. BOEHM *et al.* note that:

304 BRIAND *et al.* (1999).
305 MUKHOPADHYAY *et al.* (1992).
306 FINNIE *et al.* (1997).
307 FORRESTER (1961); FORRESTER (1968); FORRESTER (1969).
308 MADACHY (1994).
309 BOEHM *et al.* (2000a); CHULANI (1999); MADACHY (1994).
310 ABDEL-HAMID, MADNICK (1991), p. 76

> *"This is a significant departure from the other techniques […], which tend to rely on static models and predictions based upon snapshots of a development situation at a particular moment in time. However, factors like deadlines, staffing levels, […] etc., all fluctuate over the course of development and cause corresponding fluctuations in the productivity of project personnel. This in turn has consequences for the likelihood of a project coming in on schedule and within budget – usually negative."*[311]

Inspired by ROBERTS' previous work on the dynamics of R&D projects, ABDEL-HAMID and MADNICK took on the idea of System Dynamics and developed an integrative model of software development projects, which covers 1) human resource management, 2) software production, 3) controlling, as well as 4) planning.[312] This model is capable of estimating effort, its distribution over time, schedules, as well as defect rates.

Similarly, MADACHY used System Dynamics to model inspection-based software life-cycle processes.[313] With reference to MADACHY'S work, BOEHM *et al.* briefly describes a System Dynamics model that simulates BROOKS' Law, which states that *"adding manpower to a late software project makes it later."*[314]

## 2.4 Summary of Estimation Techniques and their Adoption in Practice

The previous sections 2.2 and 2.3 have shown that it is important to distinguish between *Size Estimation* and *Effort Estimation*. Tab. 2.10 provides an overview of the primary size and effort estimation techniques and their central characteristics.

**Tab. 2.10:** Summary of prominent size and effort estimation techniques

| Technique | Focus | Major input parameter | Complexity assessment | Further factors |
|---|---|---|---|---|
| Function Point Analysis | Size | Functional requirements | Low, Average, High | Value Adjustment Factor (14 general system characteristics) |
| COSMIC FFP | Size | Functional requirements | - | - |

[311] BOEHM *et al.* (2000a), p. 197.
[312] ABDEL-HAMID, MADNICK (1991); ROBERTS (1981).
[313] MADACHY (1994).
[314] BOEHM *et al.* (2000a); BROOKS (1975).

| **Technique** | **Focus** | **Major input parameter** | **Complexity assessment** | **Further factors** |
|---|---|---|---|---|
| Object Points | Size | Screens, reports, modules | Simple, medium, difficult | Productivity rate for simple effort estimation |
| Use Case Points | Size | Use Cases | Influences of a set of differently weighted factors | Technical & Environmental Complexity Factor, Productivity Factor |
| COCOMO | Effort | Lines of Code (KDSI) | - | Development mode, Effort Adjustment Factor (15 cost drivers) |
| COCOMO II | Effort | Lines of Code (KDSI) | - | Scale factors (5); Effort Adjustment Factor (17 cost drivers) |
| Expertise/analogy-based estimation | Size/Effort | Experience, historical data | Intuition, comparison | Unlimited |
| Neural Networks | Effort | Size | Overall complexity rating can be used as input data | Unlimited |

To recapitulate, development effort estimation techniques address at least one of the following three questions concerning a new software system: 1) *what* will the system be like, 2) *how* is the system going to be developed, and 3) *by whom* will the system be implemented?

Software Size is measured either in Lines of Code or abstract point-based metrics like Function Points. The Backfiring technique allows converting Lines of Code into Function Points and vice versa. The most prominent sizing techniques are IFPUG Function Point Analysis and COSMIC FFP. Besides, Object Points and Use Case Points are still in use. Generally, *sizing* techniques focus directly on the software product by asking *"what"* the system will be like. It has to be noted that most sizing techniques usually have a seamless transition from size estimation to effort estimation, since they provide simple formulas for converting size into working time.

Model-based effort estimation techniques like SLIM, COCOMO, and COCOMO II use the software size as a major input parameter for effort estimation. These techniques focus on the process of developing software by asking *"how"* and *"by whom"* the system will be developed.

Expertise-based techniques can address all three questions at once. Delphi, WBS, and Three-Point Estimation can be used to estimate size, effort, or both.

In contrast to the wide spectrum of available estimation techniques, only a few are used in practice. In a survey of 598 software development organizations, HEEMSTRA found that 60.8% use *estimation by analogy*, followed by 25.5% using *expert judg-*

*ment.*[315] Only 13.7% use *parametric models* like COCOMO in order to estimate project effort. Likewise, LEDERER and PRASAD found that the most frequent estimation approach is comparison to similar past projects based on personal memory or documented facts.[316] Additionally, they found that *intuition* and even *guessing* are favored over complex, statistical methods. In a review of studies on expert estimation of software development effort, JORGENSEN summarizes:

> *"HIHN and HABIB-AGAHI found that 83% of the estimators used 'informal analogy' as their primary estimation techniques, 4% 'formal analogy' (defined as expert judgment based on documented projects), 6% 'rules of thumb', and 7% 'models'. [...] A survey conducted in New Zealand, PAYNTER reports that 86% of the responding software development organizations applied 'expert estimation' and only 26% applied 'automated or manual models' (an organization could apply more than one method). A study of the information systems development department of a large international financial company, HILL et al. found that no formal software estimation model was used. JORGENSEN reports that 84% of the estimates of software development projects conducted in a large Telecom company were based on expert judgment, and KITCHENHAM et al. report that 72% of the project estimates of a software development company were based on 'expert judgment'."*[317]

Studies on *estimation accuracy* also favor expertise- or analogy-based techniques. The work of SHEPPERD and SCHOFIELD, MUKHOPADYAY *et al.,* and FINNIE *et al.* found higher estimation accuracy for *estimation by analogy* in contrast to algorithmic approaches.[318] For example, SHEPPERD and SCHOFIELD found that *estimation by analogy* provides higher estimation accuracy than approaches based on stepwise regression analysis.[319] In this study, accuracy was determined by the *Mean Magnitude of Relative Error* (MMRE) as well as the *Pred(25)* indicator, which measures the percentage of predictions falling within 25 percent of the actual value.[320] Only Neural Networks were able to outperform analogy-based effort estimation in one study.[321]

> *"A striking pattern emerges in that estimation by analogy produces a superior predictive performance in all cases when measured by*

[315] HEEMSTRA (1992).
[316] LEDERER, PRASAD (1993).
[317] JORGENSEN (2004), p. 39. References in quotation: HIHN, HABIB-AGAHI (1991); HILL *et al.* (2000); KITCHENHAM *et al.* (2002b); PAYNTER (1996).
[318] FINNIE *et al.* (1997); MUKHOPADHYAY *et al.* (1992); SHEPPERD, SCHOFIELD (1997).
[319] SHEPPERD, SCHOFIELD (1997).
[320] *MMRE* and *Pred(25)* are explained in Section 4.3.5.
[321] FINNIE *et al.* (1997).

> *MMRE and in seven out of nine cases for the Pred(25) indicator."*[322]

However, accuracy surveys must be critically reviewed. KITCHENHAM *et al.* argue that accuracy indicators are often mistakenly applied within the research community.[323]

Another problem of estimation techniques is addressed by KEMERER. [324] Based on an empirical validation of effort estimation models, he found – besides insufficient accuracy – that none of the validated models adequately reflects those factors that are supposed to largely influence project performance.[325]

Although there is a variety of different estimation approaches, practitioners seem to favor informal techniques. Estimation by analogy or expertise dominates the field.[326] Moreover, empirical studies found that these informal approaches are even more accurate than formal model-based techniques.[327] Accordingly, JORGENSEN concludes that

> *"There is no substantial evidence supporting the superiority of model estimates over expert estimates. There are situations where expert estimates are more likely to be more accurate, e.g., situations where experts have important domain knowledge not included in the models or situations when simple estimation strategies provide accurate estimates."*[328]

Finally, HEEMSTRA gives a further supportive argument for expertise-based estimation techniques:

> *"Software cost estimation is often wrongly regarded as a technical problem that can be solved with calculation models, a set of metrics and procedures. However, the opposite is true. The 'human aspects' are much more important. The quality, experience and composition of the project team, the degree in which the project leader can motivate, kindle enthusiasm, and commit his developers,*

322 SHEPPERD, SCHOFIELD (1997), p. 742.
323 KITCHENHAM *et al.* (2001).
324 KEMERER validated SLIM, COCOMO, Function Points, and ESTIMACS.
325 KEMERER (1987).
326 JORGENSEN (2004).
327 FINNIE *et al.* (1997); MUKHOPADHYAY *et al.* (1992); SHEPPERD, SCHOFIELD (1997).
328 JORGENSEN (2004), p. 55.

*have more influence on delivering the software in time and within budget than the use of rigid calculations."*[329]

To sum up, research on effort estimation is driven by *two* primary goals:

1) *Developing new or improving existing effort estimation techniques* in order to make them capable of reflecting all relevant factors of a project and a software system. However, studies on the use of expert estimation in practice suggest that practitioners have a preference for simple, informal, and open techniques. Such techniques, e.g., estimation by analogy or intuition, are not restricted to a set of factors. Moreover, these techniques allow answering all three questions (*"what, how, and by whom"*) at once.

2) *Improving estimation accuracy*. Again, studies have found that the least formal techniques, e.g., estimation by analogy or expertise, provide the highest estimation accuracy. This suggests a gap between the potential accuracy of an estimation technique and how it is actually implemented in practice.

However, a *third* goal emerges when considering the aforementioned discussion of effort estimation and especially when taking HEEMSTRA'S supporting argument for expertise-based estimation techniques in account. That is, effort estimation techniques must respect the *human aspects* of a development project, which is in line with the suggested sociological nature of software development. Managers might favor expertise-based techniques because they leave room for such aspects. For example, project managers can instinctively incorporate the personality of the project leader, his or her relationship to the assigned developer team, and the individual capabilities and experiences of each developer. There might be important factors like enthusiasm and motivation that are not reflected by the presented estimation techniques.

Accordingly, the next chapter analyzes the nature of software development in order to determine, whether the consideration of human aspects is of utmost importance in development projects, or whether the state-of-the-art, especially the industrialization of software development, refuses this third goal, so that the consideration of people is extraneous and irrelevant.

[329] HEEMSTRA (1992), p. 638.

# 3 THE CRUX OF SOFTWARE DEVELOPMENT AND ITS RELATIONSHIP TO EFFORT ESTIMATION

What is software development? With concern to this question, *technologists* would speak about programming languages, compilers, application frameworks, development environments, conceptual design approaches, and a never-ending armada of useful tools. In opposition, *sociologists* would speak about people first. DEMARCO and LISTER would agree upon the latter, since they argue that *"the major problems of [software development] are not so much technological as sociological in nature."*[330] The research community of software engineering, however, comprises an ideological camp that wants software development to be a purely technological challenge. Accordingly, researchers as well as practitioners have constantly asked for an industrialization of software engineering over the past decades. In 1968, on the *NATO Conference of Software Engineering*, DOUGLAS MCILROY was one of the first to ask for *industrialism* in the field of software engineering:

> *"We undoubtedly produce software by backward techniques. [...] Software production today appears in the scale of industrialization somewhere below the more backward construction industries. [...] I would like to investigate the prospects for mass-production techniques in software. [...] What I have just asked for is simply industrialism, with programming terms substituted for some of the more mechanically oriented terms appropriate to mass production."*[331]

MCILROY especially asked for software components and standard catalogues of routines that are classified in terms of robustness, precision, performance etc. From today's perspective, his claim headed towards standardized code libraries, software reuse, and component-based development. The idea of industrializing software development has persisted until today as software development still relies on the

[330] DEMARCO, LISTER (1987), p. 4.
[331] DOUGLAS MCILROY (1968), p. 79.

craftsmanship of skilled individuals.[332] GREENFIELD and SHORT summarize what industrialization means for other industries:

> *"When faced with similar challenges many years ago, other industries moved from craftsmanship to industrialization by learning to customize and assemble standard components to produce similar but distinct products, by standardizing, integrating and automating their production processes, by developing extensible tools that could be configured to automate repetitive tasks, by developing product lines to realize economies of scale and scope, and by forming supply chains to distribute cost and risk across networks of highly specialized and interdependent suppliers."*[333]

Until now, the software industry has not been industrialized, and we neither know when it will happen, nor what it will be like.[334] For that reason, this chapter focuses on the fundamental question whether the software industry can be industrialized at all. By looking at other industries, this chapter illustrates the *crux* of software development, and gives arguments whether or not the software *industry* lives up to its name.

Accordingly, this chapter focuses on the nature of software development from a predominantly *economic* and *managerial* perspective. Its emphasis is on *cost-effectiveness.* The goal of this discussion is to explain what software development *is*, and what it *is not*. Without making clear the underlying understanding of software development, one might retort that particular arguments and thoughts will become unfounded, when the software industry becomes *industrialized*, or when organizations move away from *sequential* to *agile* development. Therefore, this chapter defines the economic possibilities and limitations of the software industry, and gives reasons why software development might generally be a nightmare for managers who depend on perfect control, stable project planning, and homogeneous developer teams.

Finally, the crux of software development is connected to the main subject of this thesis in order to explain its relationship to effort estimation.

---

332 GIBBS (1994); GREENFIELD, SHORT (2003).

333 GREENFIELD, SHORT (2003), p. 17.

334 GREENFIELD, SHORT (2003).

## 3.1 The Software Crisis Revisited

Simultaneously to the introduction of *"Software Engineering"* [335] in the late 1960s, the term *"Software Crisis"* emerged which originally addressed

> *"[...] the problem of building large, reliable software systems in a controlled and cost-effective way."*[336]

Software development was limited to small projects at that time, and the early attempts to build large software systems were likely to fail.[337] Software was typically written in *COBOL*, *FORTRAN,* or *Assembler* on machines that we would not call computers from today's perspective.[338] In order to understand the situation of the late 1960s, one must recall how software or programs, respectively, were created in those days. Being a programmer was not a profession. Programmers could not use high level programming languages or integrated development environments. Computers – if at all – had tiny monochrome displays. Likewise, computer keyboards were not a standard. Thus, programs were usually entered via punched cards. It was developed in *build-and-fix* or in simple *waterfall* processes.[339]

> *"[...] the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem. [...] As the power of available machines grew by a factor of more than a thousand, society's ambition to apply these machines grew in proportion, and it was the poor programmer who found his job in this exploded field of tension between ends and means. The increased power of the hardware, together with the perhaps even more dramatic increase in its reliability, made solutions feasible that the programmer had not dared to dream about a few years before. And now, a few years later, he had to dream about them, and, even worse, he had to transform such dreams into*

335 NAUR, RANDELL (1969).
336 KRUEGER (1992), p. 132.
337 RANDELL (1979).
338 DIJKSTRA (1972).
339 BERRY (2004); SCHACH (1992). The *build-and-fix* or *code-and-fix* model describes a basic development cycle: 1. Build the first version of the software. 2. The software is modified until the client is satisfied. 3. The software is used until problems occur. When problems occur, go back to step 2.

> *reality! Is it a wonder that we found ourselves in a software crisis?"*[340]

However, since that time the field of software engineering has been evolving. It has been influenced and improved by lots of noteworthy contributions and advancements.[341] Some of these contributions are briefly presented in the following in order to outline what has changed since the late 1960s.

In 1972, the first 8-bit microprocessor was released. From that moment on, computers became faster, smaller, and affordable for nearly everyone. Computers became part of Ethernet networks for the first time around 1975.[342] 16 years later, IBM entered the personal computer market and coined the term "*IBM-compatible*."[343] Punched cards have been completely replaced by terminals, keyboards, and magnetic disks. Simultaneously, operating systems became much more sophisticated. In 1969, the first version of *UNIX* was released. Personal computers were usually delivered with *MS-DOS* that appeared in 1981.[344] Four years later, the first version of Microsoft *Windows* was released. In 1991, LINUS TORVALDS published the first version of the open-source operating system *Linux*.

Since the late 1960s, many new programming languages and techniques have been developed. In 1968, the first publications on structured programming as a new programming paradigm were published.[345] In the same year, DONALD KNUTH published the first volume of *"The Art of Computer Programming."*[346] In 1972, the programming language *C* was published.[347] Seven years later, *C++* as one of the most influential *object-oriented programming* languages followed.[348] Although the foundations of object-oriented programming were invented in the late 1960s, object-oriented programming did not become popular until the 1990s. In this decade, *Java* was released offering platform independency by using a virtual machine.[349]

Other innovations have changed and standardized the way data was accessed and managed. In 1970, the *relational database model* was proposed by CODD.[350] The

[340] DIJKSTRA (1972), p. 861.
[341] A comparable, more detailed overview of the last four decades of software engineering is given by BOEHM (2006).
[342] METCALFE, BOGGS (1976).
[343] *IBM (abbr.): International Business Machines (Corporation).*
[344] *MS-DOS (abbr.): Microsoft Disk Operating System.*
[345] DIJKSTRA (1968).
[346] KNUTH (1968).
[347] KERNIGHAN, RITCHIE (1978).
[348] STROUSTRUP (2000).
[349] SUN MICROSYSTEMS INC. (2008).
[350] CODD (1970).

fundamental theories on *database normalization* followed in the mid-1970s.[351] The *Structured Query Language* (SQL) was published in 1974.[352] The first commercial database products appeared in 1979.

As a final point, the first *Web Browser* and the *Hypertext Markup Language* were released in 1991.[353] The public became aware of the Internet and the *World Wide Web*. The dominant programming language for web clients, *JavaScript,* followed in 1995. In the same year, *PHP* as one of the most popular scripting languages for web-based applications was published.[354] Web-clients offered a completely new way of application development.

When applying MOORE's Law, which states that processing power is doubling approximately every two years, today's computers are 1,000,000 times faster than they were in the late 1960s.[355] Similarly, today's memory and disk storage capacities are 1,000,000 times higher than 40 years ago. Today's developers can use a variety of programming languages, new programming paradigms, integrated development environments, application frameworks, concurrent versioning systems, database systems, open-source software, standardized formats, and protocols.

In comparison the late 1960s, the circumstances in which software is developed today are totally different. Therefore, the *software crisis* must be deemed to be over. Its basic message, however, has persisted over the last four decades. The challenge of building *reliable* software in a *controlled* and *cost-effective* way still exists today:[356]

> *"Despite its critical importance, software remains surprisingly fragile. Prone to unpredictable performance, dangerously open to malicious attack, and vulnerable to failure at implementation despite the most rigorous development processes, in many cases software has been assigned tasks beyond its maturity and reliability."*[357]

In the following, *reliability*, *control*, and *cost-effectiveness* are examined in more detail in order to explore how each of these perspectives evolved and how they influence today's software development. The emphasis lies on cost-effectiveness as it reflects the economic perspective.

---

351 ARMSTRONG (1974); BEERI, BERNSTEIN (1979); CODD (1972); CODD (1974).
352 CHAMBERLIN, BOYCE (1974).
353 W3C (2004).
354 THE PHP GROUP (2008). *PHP (abbr.): Hypertext Preprocessor.*
355 MOORE (1965).
356 E.g., BOEHM, BASILI (2000).
357 BOEHM, BASILI (2000), p. 27.

### 3.1.1 Reliability

Following PHAM, *"reliability is defined as the probability of success or the probability that a system will be perform its intended function under specified design limits. Mathematically, reliability R(t) is the probability that a system will be successful in the interval from time 0 to time t."*[358] Reliability can be understood as a quality criterion of software systems, since *"an important quality attribute of a computer system is the degree to which it can be relied upon to perform its intended function."*[359] Moreover, reliability interacts with development costs, and, therefore, with the cost-effectiveness of software development projects:

> *"The interactions between software costs and the various software qualities (reliability, ease of use, ease of modification, portability, efficiency, etc.) are quite complex, as are the interactions between the various qualities themselves. [...] A project can reduce software development costs at the expense of quality but only in ways that increase operational and life-cycle costs."*[360]

Accordingly, the effort of ensuring a high reliability by perfectly complying with the system's specification while eliminating all errors and bugs extensively increases development costs. In addition, the difficulty to ensure reliability increases with the system size.

BELADY and LEHMAN observed the phenomenon of growth of errors in software systems that are continually modified.[361] Their mathematical description of the phenomenon, published in 1976, states that the number of errors declines in early releases until the number of errors reaches a minimum. At this point, the internal structure of the software has become decayed so that changes are more likely to cause new errors and side effects. The number of errors is increasing with new releases (see Fig. 3.1). This effect is called the BELADY-LEHMAN upswing, which negatively affects the reliability of software systems. [362]

358 PHAM (1999), pp. 13-14.
359 GOEL (1985), p. 1411.
360 SELBY (2007), p. 156.
361 BELADY, LEHMAN (1976); BELADY, LEHMAN (1985).
362 BELADY, LEHMAN (1976); BELADY, LEHMAN (1985).

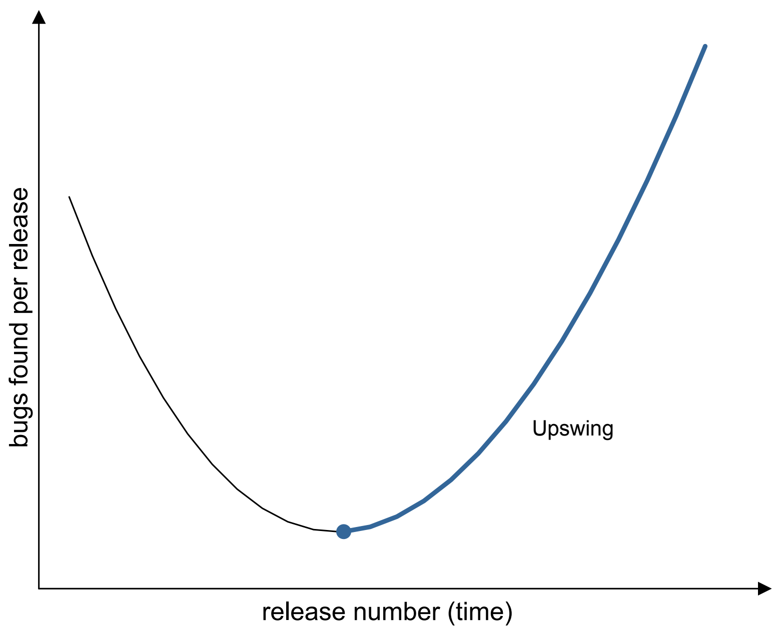


**Fig. 3.1:** The BELADY-LEHMAN graph[363]

A recent, experimental study conducted by SMIDTS *et al.* demonstrates the challenges of producing reliable software.[364] Two independent teams, using different software development methodologies, had to develop identical software systems. Although the system was simple, both teams were unable to reach a reliability level of 98%.

Likewise, in 1987, BROWN and GOULD conducted a study on the development of electronic spreadsheets using *Lotus 1-2-3*.[365] Although all participants were experienced spreadsheet developers, the study showed that 44% of 27 spreadsheets contained nontrivial errors. However, the developers rated their work as accurate and error-free.

The past, presence, and most probably the future of software reliability can be adequately pinpointed by the 9th lesson described by KANER *et al.* on lessons learned in software testing:

*"You will not find all the bugs."*[366]

---

363 BELADY, LEHMAN (1976); BELADY, LEHMAN (1985); BERRY (2004).
364 SMIDTS *et al.* (2002).
365 BROWN, GOULD (1987).
366 KANER *et al.* (2002), p. 6.

## 3.1.2 Control

### 3.1.2.1 The Era of Sequential Software Development

In 1970, ROYCE primarily documented a sequential development process model, which has become known as the *waterfall model*.[367] Although ROYCE is often cited in this context, his work was not a proposal but an early criticism on the sequential development approach. When moving away from one-man build-and-fix development towards team-based projects in the 1950s and 1960s, organizational management was confronted with software development issues for the first time. The waterfall model can be understood as a natural approach to divide labor and to give control over development processes to management. The overall process is separated into sequential phases. Each phase has to be completed before the project proceeds to the next phase (see Fig. 3.2).

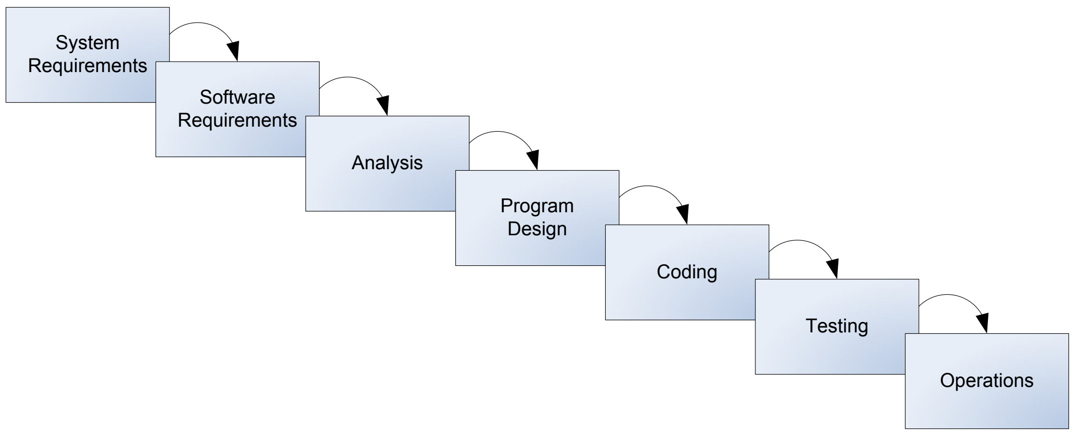


**Fig. 3.2:** Waterfall model of the software process[368]

The waterfall approach allows a straightforward integration with project management, since it suggests structure, discipline, and control over progress by providing documentations and milestones at the end of a phase. Another economic motivation of the waterfall model is an early identification of conceptual errors. MCCONNELL, who also criticizes[369] the sequential approach, gives a supporting argument:

> *"A requirements defect that is left undetected until construction or maintenance will cost 50 to 200 times as much to fix as it would have cost to fix at requirements time."*[370]

[367] ROYCE (1970).
[368] E.g., ROYCE (1970).
[369] MCCONNELL (2004).
[370] MCCONNELL (1996).

A similar rule-of-thumb has been published by BOEHM in 1987:

> *"Finding and fixing a software problem after delivery is 100 times more expensive than finding and fixing it during the requirements and early design phases."*[371]

This insight has been the major economic driver for "*Big Design Up Front*" and waterfall processes. However, in 2001, BOEHM and BASILI expect "*the cost-escalation factor for small, noncritical software systems to be more like 5:1 than 100:1.*"[372] Nevertheless, while these rules apply for requirements *defects*, the waterfall approach completely ignores requirements *changes*. According to JACKSON, only two things are known about requirements: they will change and they will be misunderstood.[373] Thus, PARNAS and CLEMENTS discuss the irrationality of a strict sequential approach:

> *"Even if we knew the requirements, there are many other facts that we need to know to design the software. Many of the details only become known to us as we progress in the implementation. Some of the things that we learn invalidate our design and we must backtrack. Because we try to minimize lost work, the resulting design may be one that would not result from a rational design process. [...] Even if we could master all of the detail needed, all but the most trivial projects are subject to change for external reasons. Some of those changes may invalidate previous design decisions."*[374]

Analogously, BROOKS comments on adopting the waterfall model:

> *"Much of present-day software acquisition procedure rests upon the assumption that one can specify a satisfactory system in advance, get bids for its construction, have it built, and install it. I think this assumption is fundamentally wrong [...]."*[375]

In 1995, BROOKS recapitulates: *"The waterfall model is wrong"* since *"the hardest part of design is deciding what to design"* and *"good design takes upstream jumping at every cascade – sometime back more than one step."*[376] Consequently, the waterfall model has been replaced in the past by innovative process models that focus on

[371] BOEHM (1987). A comparable ratio has already been described by FAGAN (1976): *"Rework [...] is 10 to 100 times less expensive than if it is done in the last half of the process."*

[372] BOEHM, BASILI (2001), p. 135.

[373] JACKSON (1994).

[374] PARNAS, CLEMENTS (1986), p. 251.

[375] BROOKS (1987), p. 12.

[376] BROOKS (1995a), slide 8.

*iterative and incremental development* (IDD).[377] The earliest references that address the concept of iteration have been published by ROYCE in 1970 and especially by MILLS in 1971 who gave a more profound view on iterative development.[378]

#### 3.1.2.2 The Era of Iterative Software Development

In 1985, BOEHM published his work on *"A Spiral Model of Software Development."*[379] In contrast to former publications, BOEHM initially formalized a risk-oriented *iterative* approach. The overall development process is divided into cycles, each of which starts with the identification of *objectives* (e.g., features or performance), *alternatives* (e.g., different designs or approaches), and *constraints* (e.g., schedule, costs, and interfaces). The focus of this assessment is to identify *uncertainties* as potential sources of project risk. Proposed approaches to resolve such risks are prototyping, simulation, benchmarking, questionnaires, and modeling. If uncertainty dominates the current project stage, the next steps will be evolutionary in order to remove uncertainty. Conversely, if most uncertainty has been successfully removed, the project will follow the basic waterfall model. Each cycle ends with a prototype and a review in order to assess the project state and to plan the next cycle (see Fig. 3.3).

[377] LARMAN, BASILI (2003).
[378] MILLS (1971); ROYCE (1970).
[379] BOEHM (1985); BOEHM (1988).

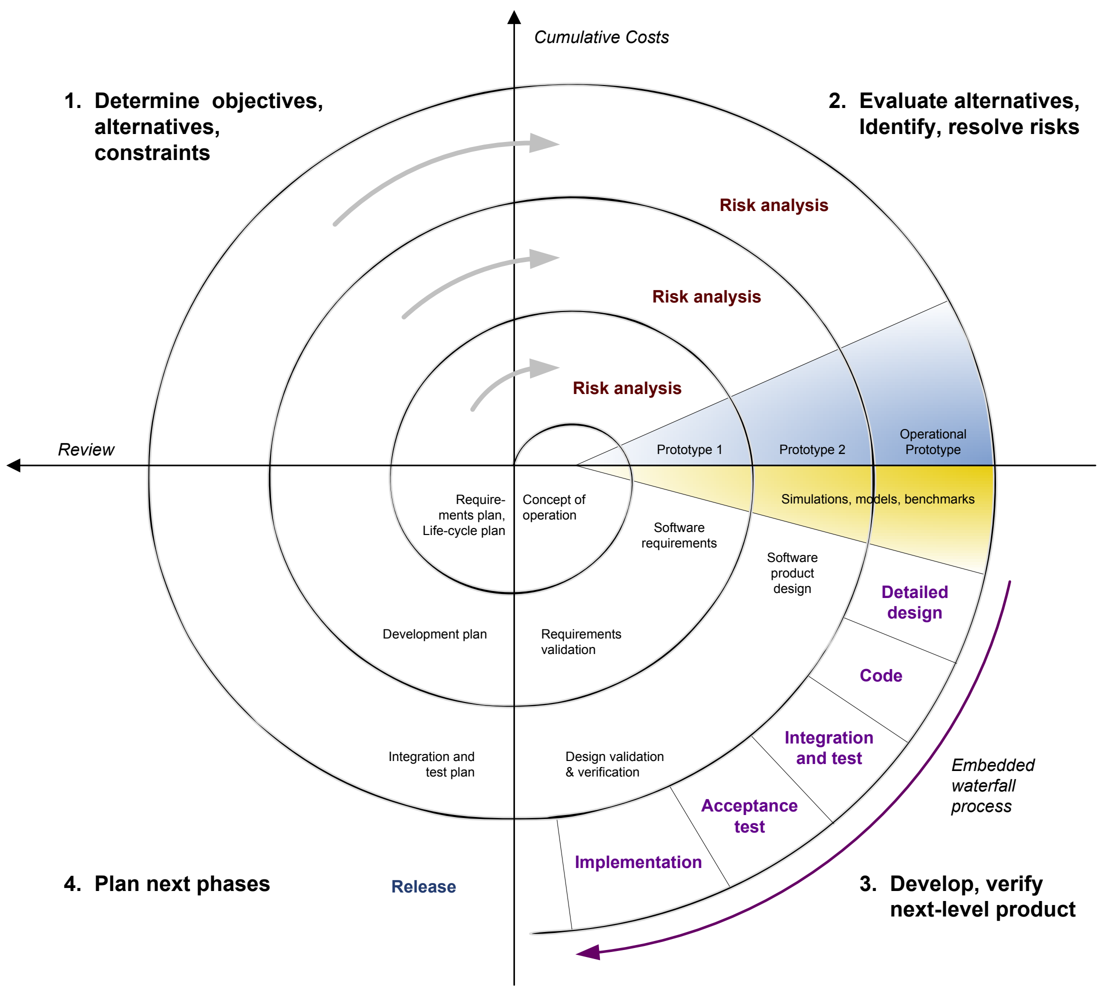


**Fig. 3.3:** Spiral model of the software development process[380]

BASILI and TURNER summarize the motivation and central concept of IDD as follows:

> *"The basic idea behind iterative enhancement is to develop a software system incrementally, allowing the developer to take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system. Learning comes from both the development and use of the system, where possible."*[381]

Iterative development emphasizes working with prototypes to measure progress. Early prototypes provoke feedback from customers and developers in order to identify changing requirements as early as possible. This approach takes account of LEHMAN's *Laws of Software Evolution*.[382] A software system (termed *e-type system*), once installed and becoming part of the application domain, alters its own require-

[380] BOEHM (1985).
[381] BASILI, TURNER (1975). Also LARMAN, BASILI (2003), p. 49.
[382] LEHMAN (1980); LEHMAN (1996).

ments. The software co-evolves with its domain and vice versa.[383] By working within short time frames or *timeboxes* respectively, iterative development methodologies incorporate this natural evolution of requirements and minimize risks during the overall development process. HIGHSMITH and COCKBURN describe the consequence for today's software project management as follows:

> *"Because we cannot eliminate these changes, driving down the cost of responding to them is the only viable strategy. Rather than eliminating rework, the new strategy is to reduce its cost."*[384]

#### 3.1.2.3 The Era of Agile Software Development

During the last decade, methodologies like *Agile Software Development*[385], *Extreme Programming*[386], *Rapid Application Development*[387], as well as *Rapid Prototyping* complemented IDD. Additionally, new management concepts like *Scrum*[388] and *Lean Software Management*[389] have been proposed. Agile software development appears to be a recent development methodology that incorporates most of the lessons learned during the past:[390] 1) Agile methods favor incremental work, 2) they focus on the talents and skills of individuals, 3) they try to reduce the cost of communication and the time between making a decision and seeing its consequences.[391] Accordingly, we must ask, how agile methods affect the control over software development processes. CORAM and BOHNER address this question by examining the impact of agile methods on software project management:[392]

- The largest impact of agile methods is on developers. They have to be team-oriented, motivated, talented, and highly communicative.[393] Thus, allocating strong developers, who fit into agile teams, becomes a major challenge for project management.
- Agile teams have *two* leader roles: *team leaders* and *project managers*. Team leaders must encourage the team to take initiative.

[383] LEHMAN, RAMIL (2001).
[384] HIGHSMITH, COCKBURN (2001), p. 120.
[385] The first agile concepts are described by EDMONDS (1974). The *"Agile Manifesto"* is written by BECK *et al.* (2001).
[386] BECK, ANDRES (2004).
[387] MARTIN (1991).
[388] The term *"Scrum"* has been initially used by TAKEUCHI, NONAKA (1986) as a metaphor. SCHWABER (1995) referenced this metaphor and described *"Scrum"* as it is used today.
[389] POPPENDIECK, POPPENDIECK (2003).
[390] HIGHSMITH, COCKBURN (2001).
[391] COCKBURN, HIGHSMITH (2001).
[392] CORAM, BOHNER (2005).
[393] CORAM, BOHNER (2005); HIGHSMITH, COCKBURN (2001).

*Leadership-collaboration* management is superior to *command-and-control* management.[394] Moreover, decision-making becomes a *shared* task within the team.[395] Again, team leaders must encourage the team to participate in decision-making processes. In contrast, project managers take responsibility for business decisions and tracking progress. However, schedules and plans are less important for agile development. Instead of focusing on a strict schedule, the project manager must ensure that the team quickly responds to change.[396] For that reason, it is common practice to arrange frequent and short meetings as it is proposed by scrum.[397] Finally, instead of relying on contracts and appointed deliverables, the project manager is responsible for establishing customer involvement and close collaboration.

- The development process is started without detailed formal planning in the beginning. Instead, the emphasis of agile methods is placed on *constant* planning. The overall process is divided into short timeboxes in order to react on upcoming changes.[398] Moreover, agile methods favor informal communication over written documentations in order to build a collective knowledge within the team.

Summarizing the above, recent development methodologies trade strict control for more flexibility and autonomy. The overall development process is not planned and scheduled upfront. Progress is made in small iterative phases while encouraging change and constant feedback. Consequently, planning becomes a permanent task. Team leadership is separated from project lead while leadership is established via collaboration. Above all, software development projects highly depend on the skills, talents, and experiences of the developers. The allocation and development of good developers is a major challenge for software project management.

## 3.1.3 Cost-Effectiveness

The third aspect, which emerged with the software crisis, is *cost-effectiveness*. When addressing the challenge of developing software at the least possible cost, the terms

[394] COCKBURN, HIGHSMITH (2001).
[395] WILLIAMS, COCKBURN (2003).
[396] CORAM, BOHNER (2005).
[397] COHN, FORD (2003); SCHWABER (1995).
[398] COHN, FORD (2003).

"*cost-effectiveness,*" "*efficiency,*" and "*productivity*" are often used synonymously in literature.

#### 3.1.3.1 The Myth of the Chaos Report

One of the most cited sources in the context of cost-effectiveness is the *Chaos Report*, primarily published in 1994.[399] The STANDISH GROUP found that 31.1% of software projects are cancelled before completion, 52.7% exceed budgets, and only 16.2% are on-time and on-budget. Moreover, the report describes an average overrun of 189% of the original *cost* estimate and an average overrun of 222% of the original *time* estimate.

JORGENSEN and MOLOKKEN doubted these findings, and carefully examined the report.[400] They especially doubt the 189% cost overrun, and give potential reasons why the results of the Chaos Report might be biased and unrepresentative: 1) STANDISH might have interpreted their own results incorrectly. A cost overrun of 189% means that the actual cost was about 2.9 times the estimated cost. It seems that STANDISH found a cost overrun of 89% instead.[401] 2) The study ignored cost *under*runs. 3) The definition of cost overrun was unusual. 4) The analyzed projects do not represent a random sample. STANDISH has been asking IT executives to *share failure stories*. Accordingly, the findings of the study are biased toward reports of failure.[402] GLASS comments on this bias:

> *"What does it mean if 70% of projects that are the subject of failure stories eventually failed? - Not much."*[403]

JORGENSEN and MOLOKKEN compare the STANDISH results with other research of the late 1980s and early 1990s. The results of JENKINS *et al.*[404], BERGERON and ST-ARNAUD[405], as well as PHAN *et al.*[406] showed cost overruns around 33%, which is clearly different from 189% (see Tab. 3.1).

[399] STANDISH GROUP INTERNATIONAL (1994).
[400] JORGENSEN, MOLOKKEN (2006).
[401] JORGENSEN, MOLOKKEN (2006).
[402] GLASS (2006).
[403] GLASS (2006), p. 16.
[404] JENKINS *et al.* (1984)
[405] BERGERON, ST-ARNAUD (1992).
[406] PHAN *et al.* (1988).

**Tab. 3.1:** Comparison of cost overrun surveys[407]

| Study | Year | Source | Country | Average Cost Overrun |
|---|---|---|---|---|
| JENKINS *et al.*[408] | 1984 | 72 IS development projects / 23 software organizations | USA | 34% |
| PHAN *et al.*[409] | 1988 | 191 software projects | USA | 33% |
| BERGERON[410] | 1992 | 89 software projects / 63 organizations | Canada | 33% |
| STANDISH GROUP[411] | 1994 | 365 IT executive managers of software projects | USA | 189% |

GLASS criticizes that lots of recent literature still cites the 1994 chaos report.[412] Publications often refer to failure rates of 70% or even 84% without noting that the reported percentages of challenged and cancelled projects have simply been summed up. Moreover, authors tend to ignore that STANDISH has repeated their studies and published new reports. These reports show a shift within the project resolutions (see Fig. 3.1). In 2000, for example, the percentage of successful projects raised from 16% to 28%.[413] The report published in 2004 shows a decline of failed projects from 31% to 18%.[414]

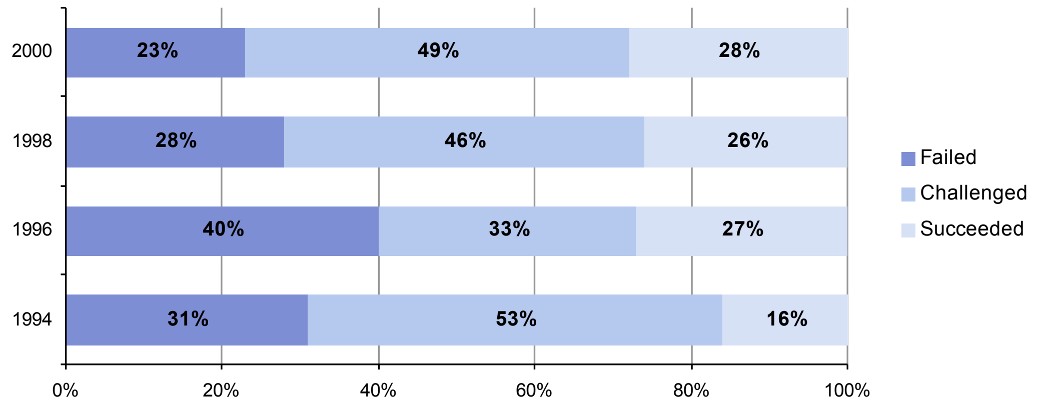


**Fig. 3.4:** Project resolutions reported by the STANDISH GROUP between 1994 and 2000[415]

Analogously, the average cost overruns have declined between 1994 and 2002. Between 2000 and 2004 the reports showed an average cost overrun of about 45%. These overruns are still higher than the results of the aforementioned academic research

407 JORGENSEN, MOLOKKEN (2006).
408 JENKINS *et al.* (1984).
409 PHAN *et al.* (1988).
410 BERGERON, ST-ARNAUD (1992).
411 STANDISH GROUP INTERNATIONAL (1994).
412 GLASS (2005).
413 GLASS (2005).
414 INFOQ.COM (2006).
415 STANDISH GROUP INTERNATIONAL (2001).

works that found average cost overruns of about 34%. However, the recent results reported by the STANDISH GROUP appear to be more plausible than the reported 189% of 1994 (see Fig. 3.5). In a review of surveys on software effort estimation, MOLOKKEN and JORGENSEN found that 60-80% of software projects have cost overruns and that the average cost overrun is about 30%.[416]

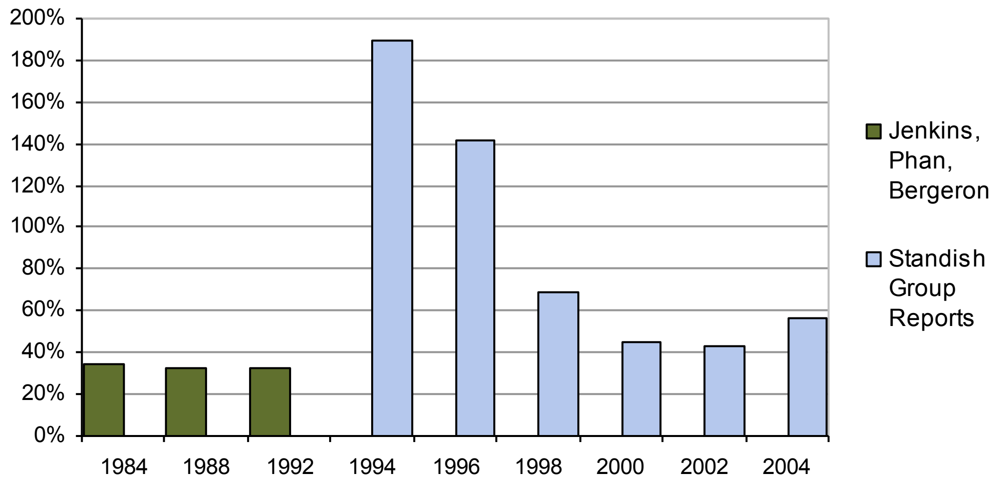


**Fig. 3.5:** Average cost overruns reported between 1984 and 2004[417]

### 3.1.3.2 The Lead of Traditional Engineering Disciplines

In the introduction of the 1994 Chaos Report, STANDISH compares software development with *bridge building*, since bridges are usually built on-time and on-budget.[418] The purpose of this introduction is to illustrate the lead of traditional engineering disciplines over software engineering. SPECTOR and GIFFORD give some insights into bridge building and explain the differences between civil engineering and software development.[419] As the most noticeable difference, SPECTOR and GIFFORD conclude that bridge design is much more structured than software design. Bridge building especially uses standardized specifications, requirements, and material constraints, which make bridges comparable.[420] The Chaos Report restricts itself to the findings of SPECTOR and GIFFORD.

In 2002, FLYVBJERG *et al.* conducted a study on *cost escalation* in transportation infrastructure projects.[421] They have collected data of 258 projects, which allowed a statistically significant study in this field for the first time. Their study covered 58

[416] MOLOKKEN, JORGENSEN (2003).
[417] Adopted from INFOQ.COM (2006).
[418] STANDISH GROUP INTERNATIONAL (1994).
[419] SPECTOR, GIFFORD (1986).
[420] SPECTOR, GIFFORD (1986).
[421] FLYVBJERG *et al.* (2002).

*rail projects* (high-speed, urban, and conventional, inter-city rail), 33 *fixed-link projects* (bridges and tunnels), and 167 *road projects* (highways and freeways). Interestingly, FLYVBJERG *et al.* found that costs are underestimated in almost 90% of the projects. Actual costs are on average 28% higher than estimated costs. Especially, rail projects have a cost overrun of 44.7% (see Tab. 3.2).

**Tab. 3.2:** Inaccuracy of transportation project cost estimates by type of project[422]

| **Project type** | **Number of cases** | **Average cost escalation** | **Standard deviation** | **Level of significance** |
|---|---|---|---|---|
| Rail | 58 | 44.7% | 38.4 | $p < 0.001$ |
| Fixed-link | 33 | 33.8% | 62.4 | $p < 0.004$ |
| Road | 167 | 20.4% | 29.9 | $p < 0.001$ |
| All projects | 258 | 27.6% | 38.7 | $p < 0.001$ |

Fixed-link projects, which refer to bridge and tunnel building, have a cost overrun of 33.8%. This percentage is similar to the reported results of JENKINS, PHAN and BERGERON (see Tab. 3.1). Moreover, looking at journals like *"Construction Management & Economics"*[423] and *"Construction Engineering & Management"*[424], one can easily find articles on cost escalation, cost overruns, and productivity issues in the field of construction.[425] Therefore, civil engineering might not be a good choice to demonstrate the lead of classic engineering over software engineering.

It is not difficult to find other cases of cost escalation in fields beyond civil and software engineering. GIELECKI and HEWLETT, for example, describe cost escalation in the field of *nuclear power plants*.[426] Between 1966 and 1977, nuclear power plants in the United States showed *overnight* cost overruns of 109% to 281%.[427] HOLLMANN and DYSERT describe cost escalations in the field of the EPC industry (Energy, Procurement and Construction). [428]

In view of that, software engineering is not exclusively affected by cost escalations. Other engineering disciplines show comparable problems in terms of cost-

[422] FLYVBJERG *et al.* (2002).
[423] TAYLOR & FRANCIS GROUP (2008). All publications in the journal of *Construction Management & Economics* have passed an anonymous double-blind review process.
[424] AMERICAN SOCIETY OF CIVIL ENGINEERS (2008). Publications in the journal of *Construction Engineering & Management* are preliminary reviewed by the journal editor. Afterwards, at least two reviewers evaluate the article.
[425] E.g., AKINTOYE (2000); ARDITI, MOCHTAR (2000); GARNETT, PICKRELL (2000); TOURAN, LOPEZ (2006).
[426] GIELECKI, HEWLETT (1994); U.S. DEPARTMENT OF ENERGY (1995).
[427] Overnight cost is the cost of a construction project without capital costs.
[428] HOLLMANN, DYSERT (2007).

effectiveness and cost estimation. As a consequence, the assumed lead of traditional engineering disciplines over software engineering becomes questionable.

#### 3.1.3.3 Optimism Bias and the Winner's Curse

A major driver for cost overruns based on inaccurate estimations refers to a psychological effect. People show a tendency to be *unrealistically optimistic* about future events. WEINSTEIN presents two studies that investigated the effect of *"optimism bias."*[429] In the first study, 258 college students estimated their own chances of experiencing predefined events. Additionally, the students estimated the corresponding chances of their classmates. WEINSTEIN found that the students significantly rated their own chances to be *above* average for *positive* events and *below* average for *negative* events ($p<0.001$).[430] The second study showed that people emphasize and overrate factors that are beneficial for their own perspective while failing to realize that others individuals may benefit from the same factors. Likewise, KLEIN and KUNDA present two related studies in which they introduce the similar *"I am above average"*-bias.[431] Translating these behavioral patterns into software development, especially inexperienced forecasters tend to make overoptimistic cost estimations, which determine future cost overruns.

FLYVBJERG *et al.* doubt that optimism bias can have a *permanent* effect on cost estimation because they suppose people learn from failure.[432] However, even if people learn from overoptimistic estimates, they will not change their behavior and continue making biased estimates. In reference to DAVIDSON and HUOT, FLYVBJERG *et al.* explain that the incentive to work with optimistic estimates is strong while the penalties for overoptimistic estimates are low.[433] They conclude that cost underestimations are best explained by *intentional* optimism bias. Forecasters and project promoters deliberately use biased and misinterpreted estimates as strategic means – in other words: they are lying.[434] Getting projects started is much more important than having accurate estimates.

The consequences of overoptimistic estimates in a competitive environment are known as the *"winner's curse"* which has been studied in other fields than software development, for example, auction theory.[435] Here, aggressive bidding in combina-

[429] WEINSTEIN (1980).
[430] WEINSTEIN (1980).
[431] KLEIN, KUNDA (1994).
[432] FLYVBJERG *et al.* (2002).
[433] DAVIDSON, HUOT (1989).
[434] FLYVBJERG *et al.* (2002).
[435] KAGEL, LEVIN (1986).

tion with biased estimates leads to low or even negative profits. JORGENSEN and GRIMSTAD adopted the winner's curse to software development.[436] They studied its effect on the basis of empirical data and a Monte Carlo simulation model. Their analysis showed that the winner's curse can be a major driver for software cost overruns and low software quality. In a group of bidders, the company that is most optimistic will win the bidding process. Assuming that the selection strategy focuses on price instead of competence, and that the group of bidders shows a large variance of optimism, it is most probable that the winner will face delays and cost overruns due to over-optimistic estimates.[437]

Fig. 3.6 illustrates how predefined cost overruns can emerge from a bidding process. In this example, seven companies (A-G) have made their bids. G has been most optimistic and wins the bidding process. The fact that G's bid is too low to make any profit is unknown *ex ante* because all bids are based on rough estimations. Since revenues are below the least possible costs, G will be confronted with cost overruns at later stages of the development process.

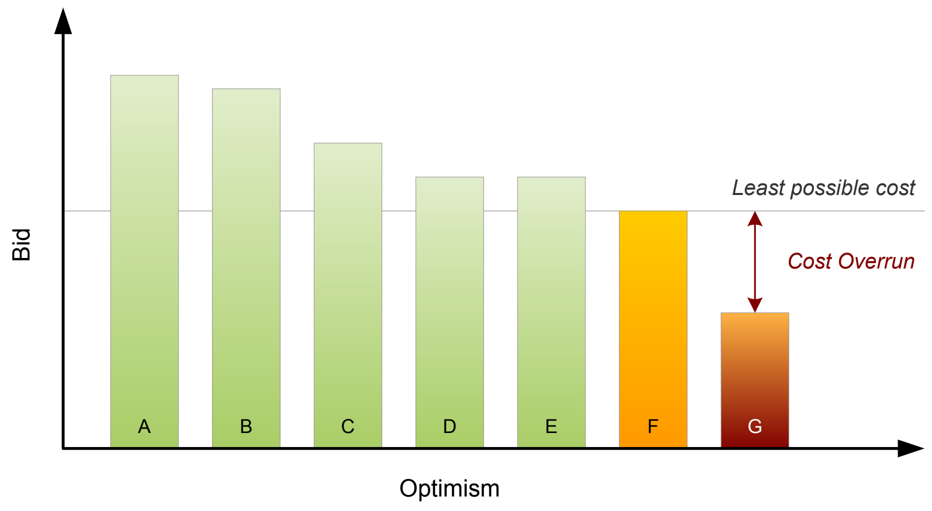


**Fig. 3.6:** Over-optimism in a bidding process

Based on a series of experiments on predicting task completion times, NEWBY-CLARK *et al.* found that participants focus on optimistic scenarios while evaluating pessimistic scenarios to be less plausible.[438] They found that pessimistic scenarios are taken into account only if participants predict someone else's completion time. According to KOEHLER and HARVEY, the level of optimism increases with the level of control.[439] A software developer, for example, tends to be over-optimistic when

[436] JORGENSEN, GRIMSTAD (2005).
[437] JORGENSEN, GRIMSTAD (2005).
[438] NEWBY-CLARK *et al.* (2000).
[439] KOEHLER, HARVEY (1997).

estimating tasks he or she is responsible for.[440] Consequently, HARVEY recommends that someone other than the person who is responsible for task completion should estimate the working time.[441]

#### 3.1.3.4 Productivity Measurement

In software development projects, costs are largely determined by working hours. Accordingly, the total duration of a development project depends essentially on the *productivity* of the developer team. In general, productivity describes the ratio of outputs to inputs. Effort measured in time, e.g., man-month or man-year, represents the input. Unfortunately, it is impossible to objectively *quantify* the output of a software development project.[442] Quantitative measures that are typically used to describe the output of a software project are *Lines of Code*, *Function Points* or variants of FP.[443] All of these metrics show insufficiencies within the context of productivity measurement.

As discussed in Section 2.2.1, Lines of Code vary with programming languages and the extent of automatic code generation.[444] Hence, it might be impossible to compare estimations or merge empirical data from different projects. Additionally, an inexperienced developer, who writes confusing, complicated, and error-prone code, tends to produce much more Lines of Code than an experienced developer. A LOC-oriented productivity measure promotes bad programming styles, because as they appear to be more productive.[445]

Function Points also reflect the size of a software system based on its functional requirements.[446] However, the method is criticized for its oversimplification of component and complexity types, its choice of weights, and the inadequate consideration of internal complexity, for example, based on difficult algorithms.[447]

In the past, several studies described statistical relationships between effort, Lines of Code, and Function Points. ALBRECHT and GAFFNEY found that Lines of Code and Function Points strongly correlate.[448] Other studies, for example, by DOLADO and

[440] JORGENSEN (2004).
[441] HARVEY (2001); JORGENSEN (2004).
[442] PREMRAJ *et al.* (2005).
[443] PREMRAJ *et al.* (2005).
[444] MATSON *et al.* (1994). The terms *Lines of Code* (LOC) and *Source Lines of Code* (SLOC) are equivalent and can both be found in literature. KLOC means one thousand (kilo) lines of code.
[445] CHATMAN (1995) gives three simple code examples which are semantically identical but have different lines of code.
[446] ALBRECHT (1979). See Section 2.2.2.
[447] SYMONS (1988).
[448] ALBRECHT, GAFFNEY JR (1983).

KEMERER, support their findings.[449] According to MATSON *et al.*, effort measured in working time has a log-linear relationship with size measured in Function Points.[450] Finally, as discussed in Section 2.2.2.2, the technique of *backfiring* allows converting Function Points into Lines of Code.[451]

#### 3.1.3.5 Productivity Studies

MAXWELL and FORSELIUS present a study on software development productivity that is based on the Finish *Experience Database*.[452] Although the study was published in 2000, the empirical data used in this study is restricted to projects that commenced before 1994.[453] This subsample comprised 206 software projects from 26 companies in Finland. In order to measure productivity, MAXWELL and FORSELIUS use the total effort in working hours as the input variable and Function Points as the output variable.[454] Additionally, lots of other factors (*classification variables*), e.g., programming language, hardware platform, business sector, and development model, have been tested for their impact on productivity. The statistical analysis shows that productivity (*FP/hour*) is highest in the manufacturing (0.337) and the retail sector (0.253). Conversely, the banking and insurance sectors show the lowest productivity (0.116).[455] The analysis of the classification variables shows that *"Company"* and *"Business sector"* have the highest impacts on productivity variance (see Fig. 3.7). Interestingly, the variables "*Programming language"* and "*CASE tools"* did not significantly account for productivity variance.[456]

[449] DOLADO (1997); KEMERER (1987).
[450] MATSON *et al.* (1994).
[451] BOEHM *et al.* (2000b).
[452] MAXWELL, FORSELIUS (2000).
[453] The article does not explicitly specify which projects have been taken into account. However, the description of the considered variables gives the required hint (Table 1; p. 81). The variable *YSTART* represents the start year of a project. Its values range from 1978 to 1994. This is confirmed by PREMRAJ *et al.* (2004).
[454] The Finish *Experience Database* uses the *Experience 2.0 method* – a variant of the Function Point Analysis - to calculate Function Points. This method takes algorithms into account. Additionally, requirements are classified by five instead of three levels of difficulty.
[455] MAXWELL, FORSELIUS (2000).
[456] MAXWELL, FORSELIUS (2000). *CASE (abbr.): Computer-aided Software Engineering.*

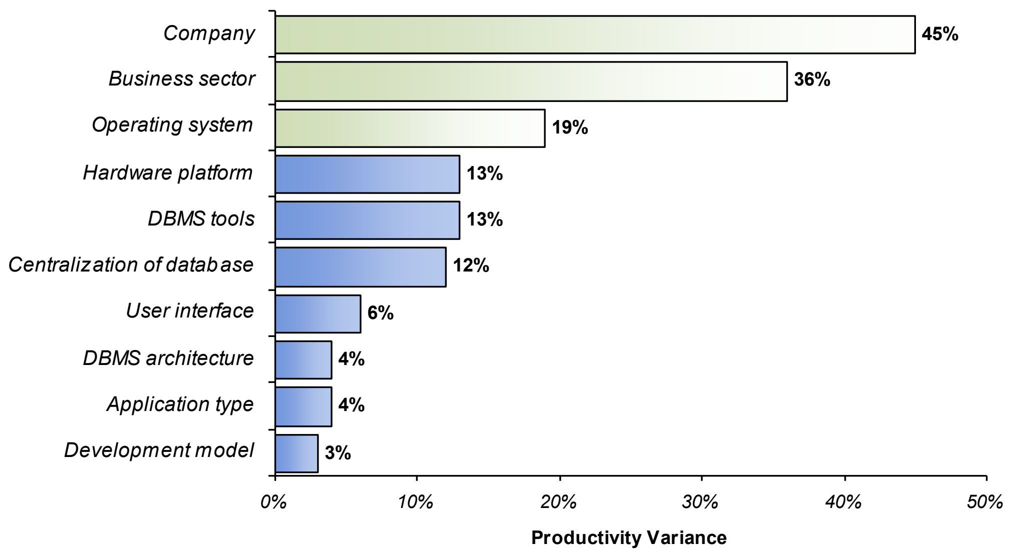


**Fig. 3.7:** Impact of classification variables on productivity variance prior to 1997[457]

The study's most interesting finding is that productivity significantly differs across companies. Unfortunately, this factor is black-boxed from the study's perspective. However, since factors like programming language, database type, and development model can only explain little productivity variance, there must be other company-specific factors that account for different productivity levels. The most obvious factor that was not used as a classification variable in the study refers to those who produce the software: the *developers*.

In 2004, PREMRAJ *et al.* published a second study based on the Finish *Experience Database.*[458] In contrast to the study discussed before, development projects prior to 1997 have been excluded from the analysis. After a further removal of outliers, the study comprises 305 projects that have been completed between 1997 and 2004. Therefore, PREMRAJ *et al.* were able to compare their results with the results of the previous study.[459]

The latest study identified the *Banking* and *Insurance* sectors as most productive (0.538 and 0.230 *FP/hour*). PREMRAJ *et al.* assume that investments into new architectures, integrated systems, and process improvements during the early 1990s have paid off for these sectors. Conversely, *Manufacturing* and *Retail* have not moved from separated applications to integrated ERP systems until the late 1990s.[460] At that time, the productivity of these sectors has dropped by 50% and even 78% (see Fig. 3.8).

[457] MAXWELL, FORSELIUS (2000).
[458] PREMRAJ *et al.* (2004).
[459] MAXWELL, FORSELIUS (2000).
[460] *ERP (abbr.): Enterprise Resource Planning.*

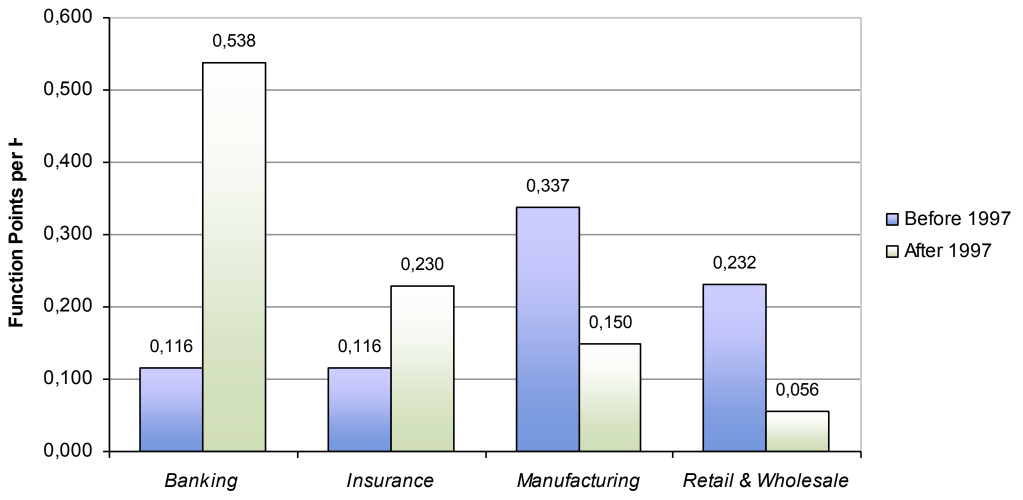


**Fig. 3.8:** Comparison of average productivity across business sectors and studies[461]

The impact of individual factors on productivity has changed between the two studies. In the latest study, the *Business Sector* showed the highest impact, followed by *Company*, *Programming Language* and *CASE Tools* (see Fig. 3.9).[462] Additionally, in comparison to MAXWELL and FORSELIUS, factors of the latest study could explain productivity variance more accurately. *Business Sector* and *Company* determined more than 75% of variance. In the former study, these factors could explain less than 50% of productivity variance (see Fig. 3.7).

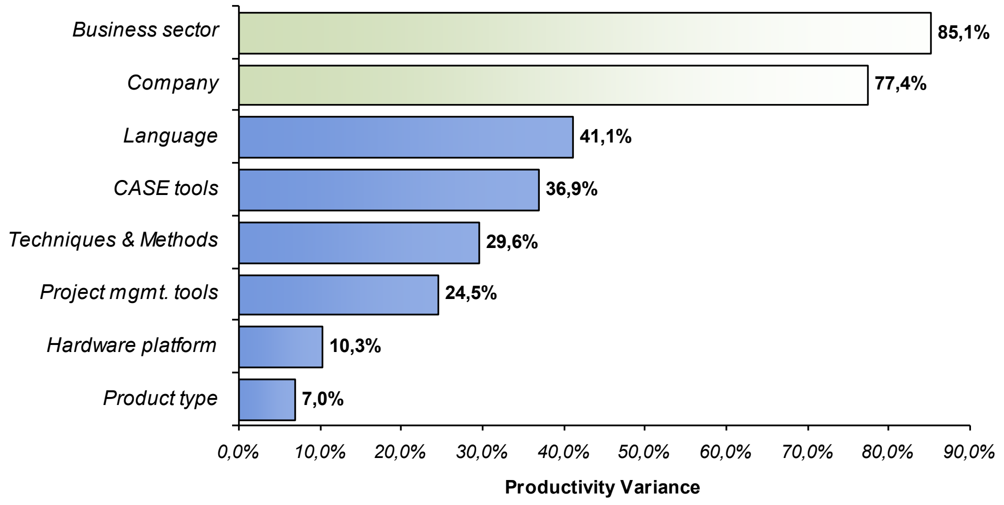


**Fig. 3.9:** Impact of classification variables on productivity variance after 1997[463]

[461] PREMRAJ *et al.* (2004).

[462] PREMRAJ *et al.* (2004).

[463] The figure is based on the data of Table 1 in PREMRAJ *et al.* (2004).

### 3.1.3.6 Productivity over Time

In a systematic review of software development cost estimation studies, JORGENSEN and SHEPPERD found that only a small number of researches show a long-term interest in software cost estimation. Lots of authors have a short-term interest in this topic, for example, as part of a PhD.[464] Accordingly, long-term productivity studies in the field of software development are rare.

In a study of *"CMM Level 5"* projects, AGRAWAL and CHARI give an overview of past research on *development effort*, *software quality*, and *cycle time*.[465] Some of these studies use *effort* as the dependent variable and *size* (measured in LOC or FP) as the independent variable (see Tab. 3.3). However, these studies either cover short time periods, disregard longitudinal productivity analyses, or they are based on a low number of observations. AGRAWAL and CHARI conclude that there is a need to reexamine the relationship between development outcomes and the major factors of influence.[466] They argue that past studies have become out-of-date due to various technological innovations and new best practices. However, the empirical data provided by AGRAWAL and CHARI refers to 34 projects that have been run between 1998 and 2004. Thus, this study once more does not focus on long-term productivity.

**Tab. 3.3:** Previous studies of cost estimation[467]

| Study | Years | Number of observations | Dependent variable | Independent variable |
|---|---|---|---|---|
| KEMERER[468] | 1981 - 1987 | 15 | Effort | Size |
| BANKER and KEMERER[469] | 1970 - 1980 | 16-63 | Effort | SLOC |
| MUKHOPADHYAY *et al.*[470] | 1981 - 1987 | 15 | Effort | Function Counts |
| MATSON *et al.*[471] | - 1994 | 104 | Effort | Size (FP) |
| MAXWELL *et al.*[472] | 1988 - 1999 | 108 | Effort | Size |
| BOEHM *et al.*[473] | - 1998 | 161 | Effort | Size |
| HARTER *et al.*[474] | 1993 - 1998 | 30 | Effort | Size |

[464] JORGENSEN, SHEPPERD (2007).
[465] AGRAWAL, CHARI (2007). *CMM (abbr.): Capability Maturity Model.*
[466] AGRAWAL, CHARI (2007).
[467] AGRAWAL, CHARI (2007).
[468] KEMERER (1987).
[469] BANKER, KEMERER (1989).
[470] MUKHOPADHYAY *et al.* (1992).
[471] MATSON *et al.* (1994).
[472] MAXWELL *et al.* (1996).
[473] BOEHM *et al.* (2000b).
[474] HARTER *et al.* (2000).

As a rare example, PREMRAJ *et al.* published an empirical, longitudinal analysis of software productivity in 2005.[475] Their analysis is based on 622 projects that were run between 1978 and 2003. Approximately 7% of the projects are maintenance projects. The empirical data is once again taken from the Finish *Experience Database.* After the removal of suspect values and extreme outliers, 602 projects remained in the edited dataset. A basic summary of the edited dataset is given in Tab. 3.4.

**Tab. 3.4:** Basic summary of the edited dataset used by PREMRAJ et al.[476]

| Variable | Mean | Median | Min | Max |
|---|---|---|---|---|
| Project Size (FPs) | 543 | 329 | 6 | 5,060 |
| Effort (person hrs) | 3,967 | 1,789 | 55 | 63,694 |
| Productivity[477] | 0.21 | 0.16 | 0.034 | 0.92 |

In order to give a rough overview of how productivity changed over time, PREMRAJ *et al.* divided the sample into two subsamples, one containing projects from 1978 to 1994 (17 years) and one containing projects from 1997 to 2003 (7 years). Some basic information of these two samples is given in Tab. 3.5. As presented before, the first subsample has already been analyzed by MAXWELL and FORSELIUS.[478] The comparison of these two subsamples shows a productivity increase of approx. 33%.

**Tab. 3.5:** Productivity comparison of 1978-1994 and 1997-2003 by PREMRAJ *et al.*[479]

| | Dataset 1 | Dataset 2 |
|---|---|---|
| Project start dates | 1978 – 1994 | 1997 – 2003 |
| Number of companies | 26 | 17 |
| Number of projects | 206 | 401 |
| Project sizes (FPs) | 33 – 3,375 | 27 – 5,060 |
| Productivity FPhr$^{-1}$ | 0.177 | 0.233 |

PREMRAJ *et al.* advice against taking the identified productivity increase of 33% for granted, since there are notable differences between the two samples. First, the distribution of projects between *business sectors* is not constant. Especially, the insurance sector has grown notably. Second, *project size* is continuously declining over time. The figures presented by PREMRAJ *et al.* suggest that the average project size

475 PREMRAJ *et al.* (2005).

476 PREMRAJ *et al.* (2005).

477 The figures are given as-is in PREMRAJ *et al.* (2005): "Effort is recorded in person hours and raw productivity is defined as the ratio of size to effort, i.e. FPs per hour." However, it is not clear how PREMRAJ *et al.* (2005) calculated the presented productivity figures. For example, 5,060 divided by 63,694 is 0.079 which is not 0.92 (FPhr$^{-1}$).

478 MAXWELL, FORSELIUS (2000).

479 PREMRAJ *et al.* (2005).

dropped from 1,000 Function Points in 1978 below 500 Function Points in 2003. Third, *maintenance projects* (approx. 42) can only be found in the second dataset.[480]

On the basis of a visual inspection of the regression plot, PREMRAJ *et al.* conclude that productivity has improved over time. They also identified a short deterioration of productivity during the mid-1990s. Further analyses showed that there is no signification difference between New Development and Maintenance projects in terms of productivity. However, Maintenance projects showed significant *economies of scale*.

Finally, PREMRAJ *et al.* identified *Company*, *Business sector*, *Year,* and *Hardware* as the most significant factors in explaining productivity (see Tab. 3.6). It has to be noted that *Process Model* has been removed from the list of explaining factors, mainly because this variable had lots of different values with many of them having a small number of observations. Moreover, a Chi-square test of *Process Model* and *Company* suggested a dependency between these two factors.

**Tab. 3.6:** Productivity factors identified by PREMRAJ *et al.*[481]

| **Variable** | **Percent of variance explained** |
|---|---|
| Company | 26.2% |
| Business sector | 11.7% |
| Year | 8.4% |
| Hardware | 5.6% |

Another promising research project on software productivity analysis is described by LIEBCHEN and SHEPPERD.[482] Their analysis is based on a dataset containing 25,000 projects of one company that is developing software solutions for numerous business sectors. After data cleaning, the dataset still comprises 1,413 finished projects that were commenced since 1990. However, there is only one paper on this research project published in 2005. At that time, the research was in progress and this particular publication merely outlines the research project.[483] Unfortunately, there is still no publication that provides final results. One reason might be that the investigated company has revoked the right to use the data for scientific publications. LIEBCHEN and SHEPPERD already discussed this risk in their paper.[484]

[480] PREMRAJ *et al.* (2005).
[481] PREMRAJ *et al.* (2005).
[482] LIEBCHEN, SHEPPERD (2005).
[483] LIEBCHEN, SHEPPERD (2005).
[484] LIEBCHEN, SHEPPERD (2005).

#### 3.1.3.7 Individual Differences

Although previous research has revealed significant differences in terms of productivity and programming performance between individuals, emphasizing developer skills is fairly new in the field of software development.[485] In a study published in 1968, SACKMAN *et al.* found productivity differences of up to 28:1.[486] Likewise, SCHWARTZ wrote: *"Within a group of programmers, there may be an order of magnitude difference in capability."*[487] In the 1970s, BOEHM characterized productivity variations of 5:1 between individuals as common.[488] In a publication on controlled experiments in program testing and code walkthroughs, MYERS described one of his observations:

> *"There is a tremendous amount of variability in the individual results. [...] The variability among student programmers is generally well known, but the high variability among these highly experienced subjects was somewhat surprising."*[489]

As presented is Section 2.3.1.2, the *Constructive Cost Model* also takes different levels of skill and experience into account.[490] Due to the mathematical formulation of the model, these levels can have a high impact on productivity and effort. BOEHM demonstrates that – all other factors being equal – a 90th-percentile team can be four times more productive than a 15th-percentile team.[491]

A detailed literature review on *people and organizations* in software production is given by NASH and REDWINE.[492] They focus on the issue of how best to select, train, organize, and manage people in view of the increasing demand for productive people in the context of software development. LAUGHERY JR. and LAUGHERY SR. present a literature review focusing on *human factors* in software engineering.[493] BEECHAM *et al.* present a systematic literature review on *motivation*.[494]

Similarly, CURTIS reviews the psychological research on software engineering that has been performed since the late 1960s.[495] In this review, he criticizes, for example,

485 GLASS (2002).
486 SACKMAN *et al.* (1968). Criticism on SACKMAN's analysis, especially on the 28:1 ratio, is given by DICKEY (1981).
487 SCHWARTZ (1968), p. 110.
488 BOEHM (1975).
489 MYERS (1978), p. 763.
490 BOEHM (1981).
491 BOEHM (1981).
492 NASH, REDWINE (1988).
493 LAUGHERY JR., LAUGHERY SR. (1985).
494 BEECHAM *et al.* (2008).
495 CURTIS (1984).

that *"the individual differences model has never been applied to programming as effectively as it should have been."*[496] CURTIS provides a set of individual characteristics, which are supposed to have a major impact on programming performance (see Fig. 3.10).

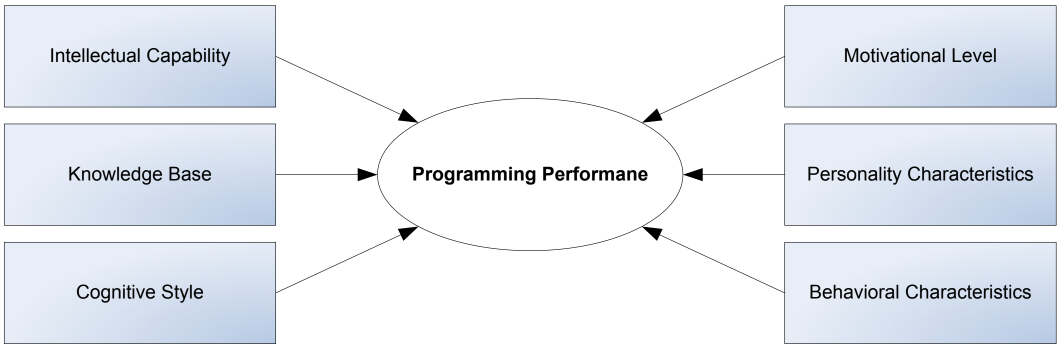


**Fig. 3.10:** Factors affecting individual programming performance[497]

The *individual differences paradigm* is frequently used in psychological studies. It is motivated by the idea that:

> *"Every man is in certain respects a) like all other men, b) like some other men, and c) like no other men."*[498]

More often than not, statistical analyses implicitly assume individual differences between study participants to be errors. For that reason, the computation and usage of averages for a group of different individuals might be mistaken.[499] Concerning IS research, the individual difference model has been previously adopted to IS studies, for example, by ZMUD as well as AGARWAL and PRASAD.[500] ZMUD used the individual differences model to analyze MIS success.[501] He concludes that:

> *"Individual differences do exert a major force in determining MIS success. It is just as apparent, however, that much remains unknown regarding the specific relationships involved and the relative importance of individual differences when contrasted with contextual factors."*[502]

[496] CURTIS (1984) p. 98.
[497] CURTIS (1984).
[498] KLUCKHOHN, MURRAY (1953), p. 53.
[499] E.g., STERNBERG, BERG (1992), p. 45.
[500] AGARWAL, PRASAD (1999); ZMUD (1979).
[501] ZMUD (1979), p. 975.
[502] ZMUD (1979), p. 975.

Another related literature review is given by BARTOL and MARTIN who discuss the management of *information systems personnel*.[503] They highlight the importance of human resources and the quality of personnel in particular.

Although we can find specific research that has paid attention to topics like individual differences and team skill in the past, these topics take a backseat in the broad field of software engineering.[504] Correspondingly, BEAVER and SCHIAVONE wrote:

> *"Despite the intuitive relationship between development team skill and software product quality, the body of software engineering research is surprisingly sparse in its coverage of the topic. [...] very few software engineering models have attempted to capture development team skill as a driving cause of the success of software projects [...]."*[505]

Analogously, BACH argues that far too much is written about processes and methods in the context of software development.[506] In his opinion, far too little attention is paid to those who write the software. BACH comments this imbalance as follows:

> *"It's relatively easy to talk about methods, especially if we can label them. More difficult to objectify is the notion of skill. Yet skill is the core issue. Any nontrivial method, performed without skill, may cause more harm than good. [...] In software projects, skill makes the world go around."*[507]

HE *et al.* studied the related phenomenon of *Team Cognition*.[508] The setting of their study implicitly addressed individual differences as well:

> *"The mere presence of individuals with diverse knowledge is an insufficient condition for a software project team to achieve quality performance. The potential value of a team can only be realized if team members utilize their unique expertise in conjunction with the knowledge of other members."*[509]

Besides individual differences manifesting in different skills and capabilities, individual personalities can also have an impact on development performance. Correspondingly, HOWARD explains:

---

503 BARTOL, MARTIN (1982).
504 BEAVER, SCHIAVONE (2006).
505 BEAVER, SCHIAVONE (2006), p. 1.
506 BACH (1995).
507 BACH (1999), p. 149.
508 HE *et al.* (2007).
509 HE *et al.* (2007), p. 262.

> *"People work in different ways dictated by their personalities. They develop individual problem solving habits and ways of thinking about and understanding the requirements of a task. Different types of problems appeal to different mentalities."*[510]

Accordingly, different work personalities can be categorized as *Deliverers*, *Prototypers*, *Perfectors*, *Producers*, *Fixers*, *Convergers*, and *Divergers*.[511] Each personality is characterized by particular, partially exclusive properties, which can be beneficial or disadvantageous depending on the situation. HOWARD argues that

> *"Understanding developer personality characteristics can help managers put together the right team. [...] If you get the right people on the project, you are more likely to achieve the right outcome."*[512]

Lastly, only a few works try to *quantify* skill, for example, by establishing an ordinal scale measurement.[513] The NASA uses an enterprise-wide *Competence Measurement System* (CMS) as a framework to quantify knowledge, skill, and experience.[514] The CMS uses an ordinal 4-tier scale to assess the capabilities of employees. Likewise, the *Constructive Cost Model* uses an ordinal 6-tier scale to define personnel-specific effort multipliers.[515] In the context of agile development, BOEHM and TURNER present a level-based classification of developer skills (see Tab. 3.7).[516]

**Tab. 3.7:** Levels of software method understanding and use[517]

| Level | Characteristics |
|---|---|
| 3 | Able to revise a method, breaking its rules to fit an unprecedented new situation. |
| 2 | Able to tailor a method to fit a precedented new situation. |
| 1A | With training, able to perform discretionary method steps such as sizing stories to fit increments, composing patterns, compound refactoring, or complex *COTS*[518] integration. With experience, can become *Level 2*. |
| 1B | With training, able to perform procedural method steps such as coding a simple method, simple refactoring, following coding standards and CM[519] procedures, or running tests. With experience, can master some *Level 1A* skills. |
| -1 | May have technical skills, but unable or unwilling to collaborate or follow shared methods. |

510 HOWARD (2001), p. 23.
511 HOWARD (2001).
512 HOWARD (2001), p. 24.
513 BEAVER, SCHIAVONE (2006).
514 NASA OFFICE OF HUMAN CAPITAL (2006).
*NASA (abbr.): National Aeronautics and Space Administration.*
515 BOEHM (1981).
516 BOEHM, TURNER (2003).
517 BOEHM, TURNER (2003).
518 *COTS (abbr.): Commercial off-the-Shelf (Software).*
519 *CM (abbr.): Change Management.*

### 3.1.4 Discussion

In view of the presented problems and advancements, the discipline of software engineering has matured, but it is still facing difficulties. Nevertheless, it does not seem appropriate to speak of a *software crisis* any longer. However, software development is still a *challenging* task and there are many ways to escalate a project.

Whether or not the *reliability* of today's software is sufficient, depends on the application domain. Unreliable software in a critical environment might cause a catastrophe. In other domains, it is sufficient to correct remaining errors in the maintenance phase. Customer involvement should help preventing conceptual errors.

*Management control* has changed, but is still feasible, effective, and might have become much more sophisticated. However, project performance is not the result of detailed upfront planning and pedantic schedules. Instead, performance depends on the skills and talents of individual staff members. Accordingly, a company should take care of its developer staff. The organizational culture should promote knowledge exchange as well as permanent learning.

Only little research is available that addresses the evolution of software development *productivity* over time. There is evidence, however, that productivity has slightly increased in the past. Nevertheless, two out of three projects exceed budgets or schedules. Although the average cost overrun is only 30%, these figures show that the risk of cost escalation still exists.

Recent research recommends emphasizing the people factor in development projects. Previous studies have found considerable productivity differences between individual developers. Typical productivity ratios discussed in the literature are 10:1 or 5:1. The individual differences paradigm addresses the issue of different skills and capabilities of individuals. So far, only little research has adopted the individual differences paradigm to IS research. However, according to the recent literature and practice, it seems that the viewpoint on people has changed since the late 1960s. While industry and research called for a more industrialized discipline due to their dissatisfaction with software development as being the craftsmanship of skilled individuals, today's development methodologies explicitly put the focus back on this craftsmanship. What seemed wrong 40 years ago is perceived as a major success factor today.

## 3.2 Economies of Software Development

After the discussion of the software crisis from today's point of view, this subsection focuses on the economies of software development in order to investigate the opportunities and limitations of *industrializing* software development.

In discussions on the potential of industrializing software development, the process of developing software might be associated and compared with production processes.[520] In such discussions, industrial production is used to describe where software development is supposed to move towards. Industrial production offers standardization, integration, off-the-shelf components, and process automation.[521] One of the major economic drivers for industrialization is taking advantage of economies of scale, typically in the form of fixed costs degression.[522] However, it is questionable if the analogy of software *development* and industrial *production* is appropriate at all.[523]

In production theory, the term *production* refers to the conversion of inputs into outputs.[524] The definition of production includes all activities that are associated with this conversion, for example, product design, packaging, and shipping. Conversely, *manufacturing* refers to the specific act of physically creating goods.[525] By definition, the output of production is *not* restricted to tangible goods, that is, *services* as intangible goods are also comprised by production theory.[526] Among other factors, production systems differ in the *type of repetition*. The major repetition types are *job* and *mass production*.[527] Taking these characteristics into account, software development can be interpreted as a form of production, and, thus, it can be subsumed under production theory. Software development is a service that creates an intangible good with effort as input and the software product as output. However, it is still to clarify, whether software development can be associated with job or mass production.

### 3.2.1 Economies of Scale in Traditional Industries

It seems that, when calling for the industrialism of software development, we implicitly refer to the mass production of tangible goods, since this special case of production theory is an illustrative example for large economies of scale. Companies that

[520] COX (1990); GREENFIELD, SHORT (2003); KILIAN-KEHR *et al.* (2007).
[521] GREENFIELD, SHORT (2003), p. 17.
[522] CORSTEN (2007).
[523] GREENFIELD, SHORT (2003).
[524] SCHNEEWEIß (1999).
[525] SCHNEEWEIß (1999).
[526] FANDEL, BLAGA (2004); HILL (1977); HOWELLS (2004); SCHNEEWEIß (1999).
[527] SCHNEEWEIß (1999).

produce mass commodities can easily benefit from scale effects in all stages of production. *Material procurement* takes advantage of large purchasing volumes by negotiating volume-based discounts, which reduces the purchase price per unit. *Manufacturing* benefits from *machine production, continuous flow production*, and *automated production*. Investments in new and efficient machinery as well as experience curve effects reduce production costs per unit. Similarly, *packaging* and *palletizing* can also be automated. Finally, goods are shipped by a third party logistics provider or, alternatively, by the company's own truck fleet. In both cases, high order volumes reduce freight costs per unit.

The illustrated example, however, depends on a physical material flow and repetitive tasks. Before a product is mass produced, both the product and the corresponding production process must be developed and designed.[528] Depending on the product, the step of *product development* can be differentiated, whether the product refers to *subjective* or *objective newness*.[529] The type of newness depends on whether the product is completely new to the market or new to the company. If the product is new to the market, the product must be completely designed from scratch. Nevertheless, product development must provide a product prototype and its production process design. From an abstract viewpoint, the step of manufacturing is then responsible for duplicating this prototype.

Product development is generally subordinated to an organization's R&D. ZENGER presents a study that empirically examines *diseconomies of scale* in R&D.[530] The study is based on responses of 912 current and former engineers who were asked via questionnaires. ZENGER found that the performance in R&D depends on the company size. Large companies generally tend to be affected by organizational diseconomies of scale. The findings of ZENGER are consistent with former research. Different research projects of different decades found that small and medium-size companies conduct R&D more efficiently than large companies.[531] This gives evidence that product development as part of R&D represents a special phase of production in which the *economies of scale* of large companies turn into *diseconomies.*

528 CORSTEN (2007).
529 CORSTEN (2007).
530 ZENGER (1994).
531 E.g., COOPER (1964); MANSFIELD *et al.* (1971); SCHERER (1965); SCHMOOKLER (1972); YEAPLE (1992).

### 3.2.2 Economies of Scale in Software Development

As mentioned before, software development can be interpreted as a form of production. Although there are some mutual characteristics with industrial mass production, software development is different in one important aspect. In opposition to tangible goods, software does not require the physical act of duplication. Software is simply copied bit by bit. This process is fast and free of cost.[532] Software development has no typical manufacturing phase which aims at duplicating a prototype. Consequently, software development neither refers to mass production nor to job production. Instead, the product prototype that results from the product development phase represents the *final good* in the context of software development. Accordingly, software development is not comparable to mass production. It is rather analogous to product development, since both disciplines face subjective or objective newness and both aim at creating new products.

This suggests that the illustrative and inspiring examples for economies of scale in industrial production do not apply for software development. Moreover, the results of previous research on R&D give evidence that it is more likely to face organizational *diseconomies* of scale in development processes (see Fig. 3.11).

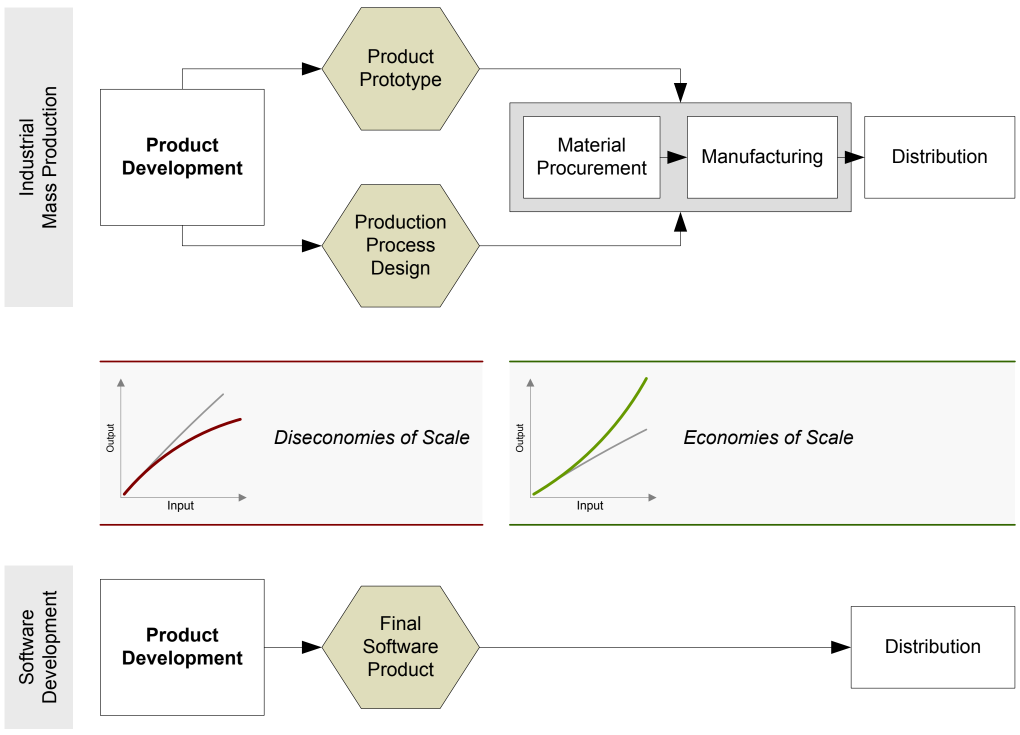


**Fig. 3.11:** Comparison of industrial mass production and software development

532 Potential costs for storage media (e.g., CD and DVD) are part of the distribution, not the duplication process.

Explicit research on economies of scale in software development has found controversial results. The analysis of the *Finish Experience Database* (602 projects) conducted by PREMRAJ *et al.* found a constant return to scale.[533] BANKER and KEMERER found that the return to scale depends on the project size allowing local scale economies as well as diseconomies.[534] They present a technique to identify the most productive scale size for companies. A later study by BANKER *et al.* based on eleven datasets confirmed the existence of both economies and diseconomies of scale in software development.[535] Commonly, such studies use the following log-linear model for the identification of economies or diseconomies of scale:[536]

$$y = a(x)^b \qquad \textbf{Eq. 3.1}$$

In this equation, *y* represents the input, usually effort measured in working hours, and *x* refers to the project size, usually measured in LOC or Function Points. In order to allow *regression* techniques, both sides of the equation are logarithmized:

$$\ln(y) = a + b\ln(x) \qquad \textbf{Eq. 3.2}$$

This equation can be directly used in linear regression models in order to estimate the factors *a* and *b*. If the estimated exponent value *b* is *less* than 1, the regression model has identified *economies* of scale. Conversely, if *b* is *greater* than 1, the regression model has identified *diseconomies* of scale. For example, the study of PREMRAJ *et al.* found *b=0.9614* for New Development Projects.[537] However, statistical tests could not confirm that the exponent is significantly different from 1, and, accordingly, the study found a *constant* return to scale.

It has to be pointed out that the conclusions of the aforementioned studies result from pure quantitative analyses. None of these studies qualitatively examines in which way a particular development technique affects the return to scale. Thus, it is important to *qualitatively* discover techniques that contribute to the return to scale. For industrial production, it is simple to imagine how economies of scale are achieved: Investing in more efficient machinery increases fixed costs while decreasing variable costs per unit. In opposition, the question which techniques account for economies of scale in software development is still unanswered.

[533] PREMRAJ *et al.* (2005).
[534] BANKER, KEMERER (1989).
[535] BANKER *et al.* (1994).
[536] BANKER *et al.* (1994); BANKER, KEMERER (1989); PREMRAJ *et al.* (2005).
[537] PREMRAJ *et al.* (2005).

### 3.2.3 Economies of Scope in Software Development

GREENFIELD and SHORT give an introduction to *software factories,* which comprise particular techniques in order to raise cost-effectiveness.[538] They argue that instead of calling for *economies of scale*, software development should look for *economies of scope.* Their view on the difference of economies of scale and scope when applied to software development reads as follows:

> *"Economies of scale arise in the production of software, as in the production of physical goods, when multiple copies of an initial implementation are produced mechanically from prototypes developed by engineers. Economies of scope arise when the same styles, patterns and processes are used to develop multiple related designs, and again when the same languages, libraries and tools are used to develop the initial implementations of those designs."*[539]

Before discussing which recent innovations are enablers for software factories, GREENFIELD and SHORT describe their vision of software factories in the future.[540] They suggest that 70% of a software product will be based on *components* including component assembly, customization, adaption, and extension. The remaining 30% are writing new code. Components will be delivered in *software supply chains,* which emerge with standard product types. Besides components, developers will use *domain-specific languages* (DSL) instead of *general purpose languages* in order to accelerate development. Moreover, *mass customization* will move into software development, allowing the on-demand customization of product variants. Finally, *organizational change* will encourage developers to think more about *assembly* instead of writing new code.

GREENFIELD and SHORT introduce three dimensions, i.e., *abstraction, granularity,* and *specificity*, in order to categorize critical innovations and to compare them with established techniques (see Fig. 3.12).[541]

---

538 GREENFIELD, SHORT (2003).
539 GREENFIELD, SHORT (2003), p. 17.
540 GREENFIELD, SHORT (2003).
541 GREENFIELD, SHORT (2003).

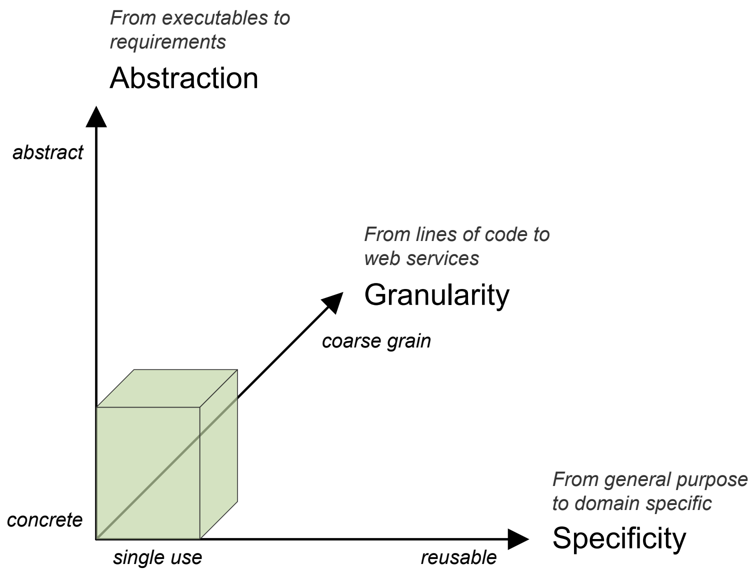

From executables to requirements
Abstraction
abstract
From lines of code to web services
Granularity
coarse grain
From general purpose to domain specific
Specificity
concrete
single use
reusable

**Fig. 3.12:** Dimensions to classify critical innovations[542]

For each dimension, GREENFIELD and SHORT describe the technological status quo, point out its major drawbacks, and present promising innovative techniques as its successors. Their point of view is briefly outlined in the following:

- First, one of the most used contemporary techniques of *abstraction* is *Object-Oriented Design and Analysis* (OOD&A). OOD&A, however, incorrectly assumes that the solution structure should match the problem structure.[543] Other drawbacks of OOD&A are the promotion of top-down development and the emphasis on visualization techniques. Instead of just visualizing information captured by models, *Model-driven Development* (MDD) aims at processing this information.[544] MDD emphasizes the transformation of models as well as the automatic code generation based on formal specifications and *domain-specific languages*.[545] These are utilized to narrow the gap between high abstraction and implementation. This gap can also be bridged by *frameworks* that raise the abstraction on the implementation level[546] or by *pattern languages* that reduce the solution complexity.[547]

[542] GREENFIELD, SHORT (2003).
[543] COPLIEN (1999).
[544] SCHMIDT (2006).
[545] VAN DEURSEN *et al.* (2000).
[546] E.g., BIGGELEBEN (2007).
[547] GREENFIELD, SHORT (2003).

- Second, software factories are supposed to increase the *granularity* of software constructs. Instead of creating 100% of a software system by manually writing new code, *component-based development* promotes the utilization of pre-built, commercial, off-the-shelf components.[548] In the past, however, the component market suffered from insufficient platform technology, which could not provide an environment for efficient, secure, and scalable interaction of large grained components.[549] This drawback might be overcome by web service technology and the corresponding idea to provide *Software as a Service* (SAAS).[550] Since the infrastructure, e.g., hardware, operating systems, application frameworks, is configured and maintained by the service provider, the service consumer does not have to take care of these aspects. In comparison to the 1990's idea of pre-built components, the integration of SAAS is supposed to cause significantly less effort. However, GREENFIELD and SHORT point out that developers must still acquire fundamental knowledge of the integrated service:

  *"It is not enough to understand how to invoke a web service component. For realistic applications, you also need to know how to perform a sequence of interactions."*[551]

- Third, *specificity* is supposed to have the strongest economic impact on software development. In the past, the software industry has stayed at low levels of specificity. Due to the economics of software development, software products favor generality in order to match as many application domains as possible.[552] With regard to economies of scope, however, the successful reuse of code, systems, and components requires a higher degree of specialization, e.g., by specializing in business sectors.

Summarizing the above, the presented vision of software factories is based on sophisticated process models like MDD, the utilization of application frameworks, domain-specific languages, pattern languages, the integration of pre-built components and SAAS, as well as a high degree of specialization. What these techniques have in

548 CHEESMAN, DANIELS (2000); D'SOUZA, WILLS (1998); GREENFIELD, SHORT (2003).
549 GREENFIELD, SHORT (2003).
550 BIGGELEBEN *et al.* (2009b); SÄÄKSJÄRVI *et al.* (2005); TURNER *et al.* (2003).
551 GREENFIELD, SHORT (2003), p. 24.
552 GREENFIELD, SHORT (2003).

common is the pursuit of economies of scope. However, before realizing scope effects, software companies must invest in the education of staff in order to raise capability and competence. Especially, specialization requires investments in domain knowledge. This again puts the spotlight on people.

## 3.2.4 Discussion

In comparison to traditional industry, software development is subject to a different form of industrialization. Economies of *scale* can only be realized during the replication of software, i.e., the *distribution* of software products. Scale effects do not apply for the *development* of software products. This point of view is also hidden in BROOKS' comments on software complexity:

> *"Software entities are more complex for their size than perhaps any other human construct, because no two parts are alike [...]. If they are, we make the two parts into one."*[553]

More explicitly, BROOKS argues:

> *"A scaling-up of a software entity is not merely a repetition of the same elements in larger size; it is necessarily an increase in the number of different elements. In most cases, the elements interact with each other in some nonlinear fashion, and the complexity of the whole increases much more than linearly."*[554]

Any piece of code is – or at least should be – *unique*. This idea is also the fundament of the DRY (*"Don't Repeat Yourself"*) programming principle.[555]

In contrast to scale effects, economies of *scope* are attainable. This field, however, is not well understood.[556] The vision of software factories gives an impression of techniques that might prepare the ground for future software development. In general, economies of scope, e.g., realized by specialization, aim at the reduction of development costs. This cost reduction implicates an increase of the productivity level. Additionally, the realization of economies of scope might suppress organizational diseconomies of scale to a certain extent. Accordingly, as the absence of scale effects prohibits exponential relations, the optimal theoretical relation of input to output is *linear*.

[553] BROOKS (1995b), p. 182.
[554] BROOKS (1995b), p. 183.
[555] HUNT, THOMAS (2000). See Section 5.1.4.1.
[556] GREENFIELD, SHORT (2003).

Fig. 3.13 visualizes four representative input/output relations in the context of software development. These relations can be explained by the evolution of an exemplary software company. Case 1 (diagram A) describes the company at a low productivity level while facing organizational diseconomies of scale. After introducing scope-oriented techniques, this company is able to suppress organizational diseconomies. This leads to a linear relation of input and output (case 2; diagram B). However, this company still starts projects from scratch. After raising staff capability and introducing an application framework, the company is able to raise general productivity. Additionally, the company is able to start projects with application prototypes spending almost no effort (positive y-axis intercept; case 3; diagram A). For completeness, case 4 (diagram B) describes an input/output relation based on economies of scale, which are regarded as unachievable in software development.

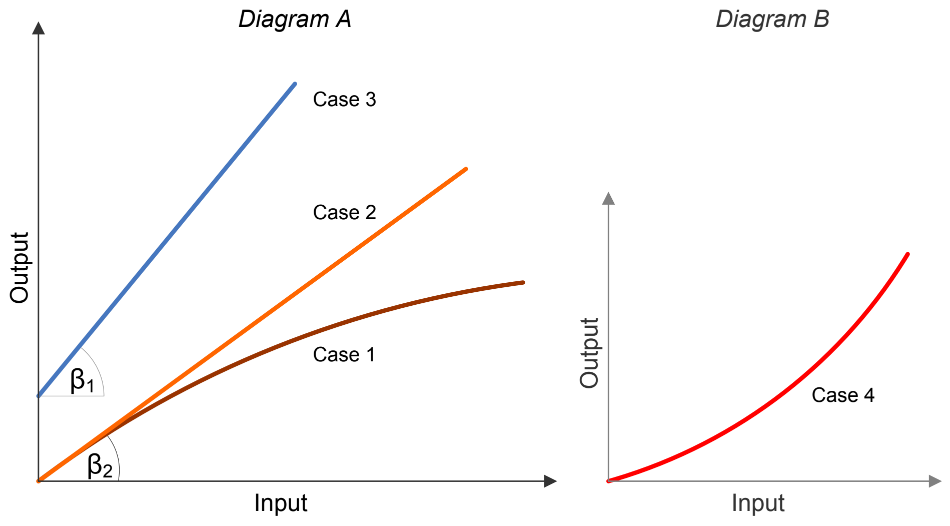


**Fig. 3.13:** Typical input/output relations in software development

The utilization of scope-oriented techniques, however, is not effortless. It demands a sophisticated understanding from developers. The vision of software factories gives strong evidence that in order to introduce, integrate, and establish scope-oriented techniques in development processes, companies must invest in specific *capability*. Therefore, software companies can realize economies of scope neither instantly nor by coincidence. Scope effects are rather a long-term, strategic objective, which requires organizational learning, explicit systematic staff training, as well as the right allocation of developers.

## 3.3 Summary

Technological progress contributes to and changes software development. Progress offers new *artifacts*, e.g., programming languages, frameworks, tools, or development paradigms.[557] These artifacts might speed up programming, or they offer completely new ways of efficient problem solution. Yet, software developers still have to combine, connect, customize, adjust, and orchestrate these artifacts. This is the *crux* of software development:

> *New technology might reduce total effort and change software development in one way or another. However, the software developer's profession is to focus on what is left. Software developers always have to address the remaining complexity and to solve the remaining problems.*

Accordingly, developers have to be aware of new technologies. They have to know how to use them efficiently. The preceding technological progress suggests that software development has become easier over time. This might be true for developers who are exceptionally talented and capable of applying each and every new technology. For less skilled developers, software development is still difficult. What good are new development techniques, if developers do not know how to use them?

With respect to the individual differences paradigm, new technologies might even widen the gap between bad and good developers, and might make matters worse. This is why software development is more sociological than technological in nature. Project managers must know who is capable of solving a particular problem efficiently. Likewise, managers must decide about the right moment to introduce and release new technologies. Although new technologies promise a reduction of effort and risk, the opposite might be true in the short term, because developers need time to learn and practice. A new technology is worthless without people who know how to apply it successfully.

Since economies of scale *do not* apply for software development, software development companies can only go for economies of scope in order to achieve a competitive advantage. However, economies of scope make high demands on staff capability. In view of that, software development has been, still is, and will be relying on the craftsmanship of skilled individuals. Industrializing software development similar to other engineering disciplines remains a wishful thinking.

According to the presented crux of software development, new technology might accelerate software development, reduce risk, and/or help identifying conceptual

[557] The term *"artifact"* is used with regard to the definition of HEVNER *et al.* (2004), p. 77: *"IT artifacts are broadly defined as constructs (vocabulary and symbols), models (abstractions and representations), methods (algorithms and practices), and instantiations (implemented and prototype systems)"*. In this section, an artifact can be anything of which developers can benefit.

design flaws. This has two consequences for effort estimation: First, new technology can reduce overall development effort. Technological progress, however, can only have its full beneficial effect if developers are capable of successfully applying new technologies. Second, the technological progress and the corresponding reduction of overall development effort do not improve estimation accuracy, since effort estimation must address the *remaining* complexity or development effort, respectively:

- The choice of development methodology, i.e., developing in sequential, iterative, or agile environments, determines development costs. This is regarded as a company-specific influence. That is, the choice has an impact on the general effort level, independent of a particular project and its specific requirements. While it does *not* improve estimation accuracy, it might help reducing project efforts in general.

- Managers can also influence the general effort level. They can promote quality, get the right people for projects, evaluate, choose, and promote appropriate development tools, programming languages, and application frameworks. They can establish quality assurance and promote rigorous testing. Besides, they are responsible for good project planning. Yet, these means are also regarded as influences on the general effort level, and, for that reason, they also help reducing general project efforts.

- In opposition to the previous two arguments, the individual differences between developers are regarded to have a project-specific impact on effort estimation. They help answering the essential question of *"Who does what?"* The challenge and major focus of effort estimation remains adequately estimating how long one developer or a one small team needs to complete a particular work package.

Accordingly, effort estimation techniques must take into account the human aspects of a development project. People are of utmost importance, and, consequently, their individual capabilities must be considered by effort estimation:

> *"The final outcome of any [software development] effort is more a function of <u>who</u> does the work than of <u>how</u> the work is done."*[558]

[558] DEMARCO, LISTER (1987), p. 93.

Motivated by the importance of the human aspects in software development projects, an exploratory study was conducted which focuses on people and their individual differences. This study addresses the question, whether it is possible to accurately estimate working effort of individuals, when there are notable differences between them. The design and results of the exploratory study as well as the gained insights are presented in the next chapter.

# 4 EXPLORATION AND QUANTITATIVE ANALYSIS OF STUDENTS' PROGRAMMING PERFORMANCE

This chapter focuses on the programming capability and performance of students. The purpose of this study is to *explore*, to what extent it is possible to estimate programming effort of individuals. After introducing the exploratory study as a research method and discussing the use of students as subjects in research studies, the empirical datasets as well as the corresponding data collection processes are described. In contrast to real-world project data, the gained dataset contains development efforts that are unaffected by estimates. Accordingly, the dataset allows analyzing estimation accuracy in a setting that is characterized by independence of actual and estimated efforts. The strategy pursued during this exploratory study is outlined at the beginning of Section 4.3.

## 4.1 Research Method

Generally, this exploratory study aims at gaining understanding of the situation, identifying interesting or controversial causal relationships, as well as getting insights and inspirations for a potential theory construction. Following SEKARAN,

> *"An exploratory study is undertaken when not much is known about the situation at hand, or no information is available on how similar problem or research issues have been solved in the past. In such cases, extensive preliminary work needs to be done to gain familiarity with the phenomena in the situation, and understand what is occurring, before we develop a model and set up a rigorous design for comprehensive investigation. In essence, exploratory studies are undertaken to better comprehend the nature of the problem since very few studies might have been conducted in that area. [...] When the data reveal some pattern regarding the phenomena of interest, theories are developed and hypotheses formulated for subsequent testing. [...] Exploratory studies are also necessary when some facts are known, but more information is needed*

> *for developing a viable theoretical framework. [...] In sum, exploratory studies are important for obtaining a good grasp of the phenomena of interest and advancing knowledge through subsequent theory building and hypothesis testing."*[559]

Likewise, BLESS *et al.* explain that

> *"The purpose of exploratory research is to gain a broad understanding of a situation, phenomenon, community or person. The need of such a study arises from a lack of basic information in a new area of interest. Most frequently, though, one must become familiar with a situation in order to formulate a problem or develop a hypothesis."*[560]

Exploratory studies can be classified as one of the following four types: 1) *theory-based* exploration, 2) *method-based* exploration, 3) *empirical-quantitative* exploration, or 4) *empirical-qualitative* exploration.[561] According to its objective, the exploratory study conducted in this research is *empirical-quantitative*.[562] Similar to *explanatory* and *confirmatory* studies, this type of exploration uses empirical quantitative datasets. The goal of an exploratory study, however, is neither explaining particular phenomena, nor testing hypotheses, but to derive *new* ideas or hypotheses from the data.[563] Besides common descriptive statistics, an approach utilizable for empirical-quantitative exploration is *Exploratory Data Analysis* (EDA).[564] In contrast to *Confirmatory Data Analysis*, EDA can be used to discover structures, trends, and patterns in quantitative data.[565] Typical techniques of EDA are *histograms*, *box plots*, *stem-and-leaf plots*, and *scatter plots*. Yet, a quantitative exploratory study is not

---

559 SEKARAN (2003), pp. 119-120.

560 BLESS *et al.* (2007), p. 47. Similar definitions of and motivations for exploratory studies are given, for example, by: KOTLER *et al.* (2006), p. 122: *"The objective of exploratory research is to gather preliminary information that will help define problems and suggest hypotheses."* STEINBERG, STEINBERG (2005), p. 53: *"When a study is exploratory, there may be little known about the topic. The goal of exploratory research is to begin to develop an understanding of the topic so that additional questions and hypotheses can be developed. Exploratory research is research at the tip of an iceberg. In other words, the researcher is trying out something for the first time."* BLANCHE *et al.* (2008), p. 44: *"Exploratory studies are used to make preliminary investigations into relatively unknown areas of research. They employ an open, flexible, and inductive approach to research as they attempt to look for new insights into phenomena."*

561 BORTZ, DÖRING (2006).

562 The objective of this exploratory study is to analyze the development efforts invested by students, and to explore, whether it is possible to accurately estimate working effort of these individuals, especially when there are notable differences between them. Accordingly, this study conducts a quantitative analysis based on an empirical dataset, and, therefore, this exploratory study belongs to the type of *empirical-quantitative* studies.

563 BORTZ, DÖRING (2006).

564 TUKEY (1977); TUKEY (1980).

565 BORTZ, DÖRING (2006).

limited to EDA. BORTZ and DÖRING argue that even tests of significance are not conflicting with exploratory studies.[566]

In this study, second semester students were used as subjects. All students attended a second semester bachelor lecture for which they had to learn and practice *SQL* as well as *Python*. Using students as subjects for experimental studies is common.[567] A recent study conducted by HE *et al.*, for example, addressed the emergence and evolution of *team cognition* in software project teams.[568] 156 undergraduate students, who had to develop a relational database system, took part in this study. Concerning using students as subjects, HE *et al.* conclude that

> *"The use of student subjects raises the possibility that findings may not accurately reflect the behavior of software project teams working in a business organization. However, most prior studies share this characteristic and there is evidence that students are good proxies for 'real-world' people in many contexts."*[569]

Nevertheless, there is an ongoing debate in IS research, whether or not students can be used as subjects for experiments.[570] SJØBERG *et al.* conducted a survey of controlled experiments in software engineering.[571] They summarize:

> *"In total, 5,488 subjects took part in the 113 experiments investigated in this survey. 87% were students and 9% were professionals. [...] The number of participants per experiment ranges from four to 266, with a mean value of 48.6. Students participated in 91 (81%) of the experiments, either alone or together with professionals and/or scientists, and professionals took part in 27 experiments (24%). The use of professionals as subjects has been relatively stable over time. Undergraduates are used much more often than graduate students. [...] Interestingly, while seven articles describe experiments using both students and professionals, only three of them measure the difference in performance between the two groups. In the first experiment, categorized as Software psychology, three programming tasks were performed. For two of the tasks, there was no difference between the groups, whereas, for the third task, the professionals were significantly better. In the second experiment, also in Software psychology, there was no difference. In*

[566] BORTZ, DÖRING (2006).
[567] SJØBERG *et al.* (2005).
[568] HE *et al.* (2007).
[569] HE *et al.* (2007), p. 287.
[570] E.g., RUNESON (2003); SJØBERG *et al.* (2005).
[571] SJØBERG *et al.* (2005).

> *the third experiment, categorized as Maintenance process, the professionals were significantly better."*[572]

Correspondingly, SJØBERG *et al.* conclude:

> *"There are good reasons for conducting experiments with students as subjects, for example, for testing experimental design and initial hypotheses, or for educational purposes. Depending on the actual experiment, students may also be representative of junior/inexperienced professionals. However, the low proportion of professionals used in software engineering experiments reduces experimental realism, which in turn may inhibit the understanding of industrial software processes and, consequently, technology transfer from the research community to industry."*[573]

In opposition to the potential lack of realism, TICHY argues that

> *"Experiments with psychology students have often been criticized for generalizing from students to the general population. University students are not representative of the general population with respect to a host of issues. Does the same criticism apply to experiments with [computer science] students? I think computer science students are much closer to the world of software professionals than psychology students are to the general population. In particular, [computer science] graduate students are so close to professional status that the differences are marginal."*[574]

To summarize, using students as subjects for experiments is common practice within the research community of software engineering.[575] Yet, research studies that use students as subjects in an experimental setting must be critically reviewed in order to detect a potential lack of realism.[576] As a result, an exploratory study bears the risk that a theory based on its findings cannot be validated or easily gets falsified if it is confronted with empirical content gained from software professionals or real-world development projects.

572 SJØBERG *et al.* (2005), p. 738.
573 SJØBERG *et al.* (2005), p. 739.
574 TICHY (2000), p. 311.
575 SJØBERG *et al.* (2005).
576 RUNESON (2003); SJØBERG *et al.* (2005).

## 4.2 Data Collection and Pretesting

Since the winter semester of 2005/06, the Chair of *Information Systems Engineering* (University of Frankfurt) has been providing a web-based e-learning platform nicknamed *"Playgrounds"* for students.[577] The first version solely offered an *SQL playground* based on the Microsoft SQL Server 2000. This playground allows students to write and test "real" SQL queries. In the following semesters, the system was extended by a *Python Playground*, a *Relational Algebra Playground*, as well as a *LINGO Playground*.[578] Each playground allows students to start learning and practicing without installing and configuring software on their private computers. Until the summer semester of 2008, the playground has accompanied eleven lectures, each attended by a range of 80 to 450 students.

### 4.2.1 Summer Semester 2008

#### 4.2.1.1 Data Collection and Data Processing

Before the summer semester of 2008, the playgrounds were extended by a logging mechanism that collects data for quantitative analyses. The extended log file structure contains the log type ("SQL" or "Python"), the client IP address, a session ID, a timestamp, the response code ("Ok" or "Error"), as well as the executed code. Moreover, the playgrounds communicate with *Google Analytics*.[579] Besides common web site analysis, Google Analytics allows geolocating IP addresses which is helpful to classify sessions geographically. An exemplary log file entry is given in Fig. 4.1.

```
[SQL] 2008-06-04 09:15:14 -- 141.2.67.115 --
200FA100DB695F01EB7D8A39EA70933B -- OK --
SELECT *
FROM fallstudie1
WHERE ItemNo = '2005' AND StoreNo < 6
---
```

**Fig. 4.1:** Exemplary log file entry of the SQL playground

Before turning to quantitative analyses, all log files have to be preprocessed. The goal of this preprocessing is merging the log files of the different playgrounds, cleaning incomplete or irrelevant data (e.g., empty codes), deriving and calculating further

[577] BIGGELEBEN, HOLTEN (2009b).

[578] *Lingo* is the name of both a programming language as well as the corresponding commercial software product developed for solving linear, nonlinear, as well as integer optimization problems. The *Lingo Playground* is used in a lecture on supply chain management, for example, to solve location and route planning problems. For further details, see LINDO SYSTEMS (2009).

[579] GOOGLE (2009). Google Analytics' feature to localize IP addresses geographically is also called *"GeoTargeting."*

metrics, and finally writing all relevant information to a database. The preprocessing was implemented in PHP.[580] The corresponding script has three stages:

1) All log files were read and processed by a parser that extracts the basic variables from the plain text log files (cp. Fig. 4.1). Moreover, during this stage, sessions of the students were "glued" together if they meet certain conditions.[581]

2) A set of metric data was derived: length of code, Lines of Code, the code level based on the HALSTEAD metric[582], the distance between subsequent codes given by the LEVENSHTEIN distance[583], a code sequence number which is based on the distance metric, and finally the time spent between executions. Additionally, the second stage identified executed codes as examples or repetitions.[584]

3) Finally, all data was written to one database table. This database table represents the *Dataset A*, which contains all cases unaggregated and almost unfiltered. The variables of this dataset are listed in the following Tab. 4.1.

580 THE PHP GROUP (2008).

581 A session, technically identified by a so-called session ID, refers to exactly one student using the playgrounds. A session usually ends if the student closes the browser window (and cookies are deleted). The data inspection, however, showed that a noteworthy number of page accesses came from identical IP addresses while having different session IDs. The analysis of the executed SQL and Python codes showed that this problem was not based upon shared proxies or routers. Instead, such page accesses came from the same students. In order to fix this problem, sessions were concatenated if they came from identical IP addresses within a short time interval. Since students share computers in the tutorial rooms, a timeout of 15 minutes was used. Accordingly, sessions were "glued" together if 1) page accesses come from the same IP address, 2) the last page access from that IP address is less than 15 minutes ago.

582 HALSTEAD (1977). The HALSTEAD metric measures code complexity. The metric reflects the difficulty to write or to understand a program. HALSTEAD found that the code complexity depends on the quantity and the ratio of used *operators*, e.g., arithmetic operators and reserved keywords like *if*, *goto*, and *while*, as well as *operands*, e.g., numeric values, strings, and variable names. Accordingly, in order to determine the difficulty of a program, three variables must be calculated: 1) the number of *distinct operators* ($n_1$), 2) the number of *distinct operands* ($n_2$), as well as 3) the *total* number of *operands* ($N_2$). The difficulty of a program is $D = (n_1/2)*(N_2/n_2)$.

583 LEVENSHTEIN (1966). The LEVENSHTEIN distance measures the difference between two strings, i.e., the minimum number of operations (insert, delete, replace) required to transform the first string into the second. For example, the strings "*Hi*" and *"High"* have a LEVENSHTEIN distance of 2 (append *g* and *h*). The strings *"Hello World"* and *"Hallo Welt"* have a LEVENSHTEIN distance of 4 (replace *e* by *a* in *Hello*, delete *o* in *World*, replace *r* by *e* in *Wrld*, replace *d* by *t* in *Weld*).

584 The playgrounds provide a set of simple examples. These exemplary SQL queries or Python programs are executed by just clicking on the corresponding links. Therefore, executed queries and programs can be identified as examples.

**Tab. 4.1:** Overview of the variables of dataset A

| **Variable** | **Description** | **Scale** |
|---|---|---|
| logType | Refers to the playground. Either "SQL" or "Python." | Nominal |
| logTime | Refers to the date and time of code execution. | Nominal |
| ip | The client IP address. | Nominal |
| sessionID | An MD5-Hash used to identify a session. | Nominal |
| responseCode | Either "Ok" or "Error." | Nominal |
| example | Either 1 (the code is an example) or 0 (otherwise). | Nominal |
| repetition | Either 1 (the exact code has been executed before during the session) or 0 (otherwise). | Nominal |
| code | Exact copy of the user input. | Nominal |
| loc | Lines of code. | Metric |
| length | Length of code (number of characters). | Metric |
| codeLevel | Code complexity (HALSTEAD metric). | Metric |
| distance | LEVENSHTEIN distance between two subsequently executed codes | Metric |
| distancePct | LEVENSHTEIN distance in percent (see above). | Metric |
| timeSpent | Time between two executions in seconds. | Metric |
| codeNo | Subsequently executed codes are supposed to belong together if the LEVENSHTEIN distance in percent is below 50%. Different code sequences are identified by the code (sequence) number. | Metric |

The playground sessions were logged from June 4th, 2008 until July 13th, 2008 (40 days). During this period, both the SQL and Python playground were exclusively used by the one lecture of interest. The lecture was accompanied by tutorials given by senior students (90 minutes; once per week). In order to attend a tutorial, students had to register via a web-based registration system.[585] In the summer semester of 2008, 405 students registered for the tutorials. The playgrounds were introduced on June 4th; the final exam was on July 14th, 2008.

The log files had a total size of 17 Mbytes with 750,025 lines representing 64,842 logged cases. Cases with empty or nonsensical codes, e.g., SQL queries executed on the Python playground and vice versa, were removed (approx. 1%). Additionally, cases, which exceeded 1,000 characters, were also removed (Python only; approx. 1%). These cases were manually identified as outliers, because these codes had been copied from an exercise document containing lots of exercise texts as comments. 95% of all cases had an execution time of less than five minutes (measured as the time between two executions). The remaining 5% of all cases ranged from five minutes to more than 24 hours. These cases are supposed to represent idle time. Con-

[585] The registration system requires a *"confirmed opt-in."* Accordingly, each student has to enter a valid e-mail address in order to receive a confirmation e-mail. The registration process is completed after clicking on a link in the confirmation e-mail. Therefore, the registration system can distinguish between pending and completed registrations. Since each student can only register once, the number of registrations is precise and accurately reflects the actual number of students in the tutorials.

sequently, these cases were removed. Finally, 59,484 cases of executed codes (Python: 31,137; SQL: 28,347) were written to the database as *Dataset A*.

As expected, the distribution of sessions per day had three peaks in the first three weeks because students attended the corresponding tutorials in which they solved exercises with the help of the playgrounds (Week 1: SQL, Week 2: Python, Week 3: Database access in Python). On the last three days before the final exam, there was another forth peak (see Fig. 4.2). On weekends and between the tutorial phase and the final exam, there were hardly any sessions per day. During the tutorial phase, 80% of all sessions came from the IP addresses of the tutorial rooms. In the last three weeks, only 15% of all sessions came from those rooms. On the last 10 days, 98.2% of all sessions came from IP addresses outside the university, i.e., the students learned at home. The geolocation report of Google Analytics confirmed that 99% of all visits came from locations within a 50 kilometer radius of the university.

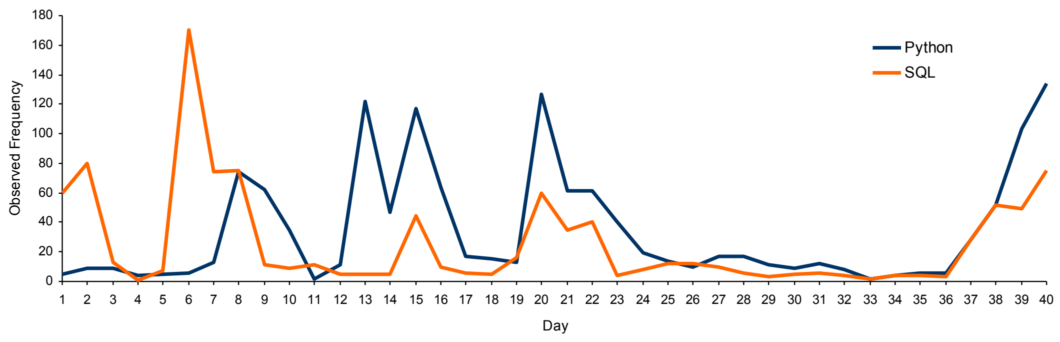


**Fig. 4.2:** Sessions per day from June 4th to July 14th, 2008 (grouped by language)

As a pretest, the summer semester dataset was subject to a preliminary statistical analysis. The findings are briefly outlined in the following.

### 4.2.1.2 Statistical Analysis

A correlation analysis based on the unaggregated *Dataset A* did not yield any *noteworthy* relationships between invested time, code length, Lines of Code, etc. In this analysis, coefficients below 0.3 are not considered noteworthy. Correlation coefficients are usually classified as "*small/weak*" (r=±0.10), "*medium/moderate*" (r=±0.30), or "*large/strong*" (r=±0.50).[586] However, there is no standardized classification or interpretation of correlation coefficients, since their interpretation depends on the given situation: *"The terms strong and weak are used to compare descriptively*

[586] COHEN (1988).

*the obtained correlation value to the value we would expect under the given circumstances."*[587]

In view of the weak correlations, cases were aggregated on the session level (*Dataset B*). Sessions with only one code execution and sessions, which solely contained examples and repetitions, were removed. *Dataset B* contains all variables of *Dataset A* in an aggregated form (totals, averages, ratios; grouped by *logType* and *sessionID*).[588] Overall, the *Dataset B* contained 2,024 cases based on 53,929 code executions.

The correlation analysis of the total time spent per session (*totalTime*), the average code level *(avgCL)*, the average code length *(avgL)*, the average number of Lines of Code *(avgLOC)*, as well as the error rate per session *(rError)* showed that neither the error rate nor one of the code metrics had a noteworthy correlation (r>0.30) with the total time (see Tab. 4.2).

However, the correlation analysis of the number of iterations (code executions per session) and the total time spent per session showed a correlation coefficient of r=0.92 (*p*<0.01). Additionally, the scatter plot of these two variables suggests a linear relationship for both SQL and Python (see Fig. 4.3).

587 WEINBERG, ABRAMOWITZ (2008), p. 130.

588 The *Dataset B* contained the following variables that could be derived from *Dataset A* using aggregation functions*: iterations* (executions per session), *codes* (number of codes per session; see *codeNo* in Tab. 4.1), *avgLOC* (average Lines of Code), *sumLOC* (total Lines of Code), *maxLOC* (maximum Lines of Code), *avgL* (average code length), *sumL* (total code length), *maxL* (maximum code length), *avgCL* (average code level), *sumCL* (total code level), *maxCL* (maximum code level), *avgDIST* (average distance), *sumDIST* (total distance), *maxDIST* (maximum distance), *sumError* (total number of erroneous executions), *rError* (error rate), *sumOK* (total number of correct executions), *avgTime* (average time per execution), *totalTime* (session duration).

**Tab. 4.2:** Non-parametric correlations (SPEARMAN)

| | | | totalTime | avgCL | avgL | avgLOC | rError |
|---|---|---|---|---|---|---|---|
| Spearman's rho | totalTime | Correlation Coefficient | 1,000 | ,015 | ,165** | ,142** | ,074** |
| | | Sig. (2-tailed) | . | ,503 | ,000 | ,000 | ,001 |
| | | N | 2024 | 2024 | 2024 | 2024 | 2024 |
| | avgCL | Correlation Coefficient | ,015 | 1,000 | ,313** | ,186** | ,191** |
| | | Sig. (2-tailed) | ,503 | . | ,000 | ,000 | ,000 |
| | | N | 2024 | 2024 | 2024 | 2024 | 2024 |
| | avgL | Correlation Coefficient | ,165** | ,313** | 1,000 | ,891** | -,034 |
| | | Sig. (2-tailed) | ,000 | ,000 | . | ,000 | ,130 |
| | | N | 2024 | 2024 | 2024 | 2024 | 2024 |
| | avgLOC | Correlation Coefficient | ,142** | ,186** | ,891** | 1,000 | -,058** |
| | | Sig. (2-tailed) | ,000 | ,000 | ,000 | . | ,009 |
| | | N | 2024 | 2024 | 2024 | 2024 | 2024 |
| | rError | Correlation Coefficient | ,074** | ,191** | -,034 | -,058** | 1,000 |
| | | Sig. (2-tailed) | ,001 | ,000 | ,130 | ,009 | . |
| | | N | 2024 | 2024 | 2024 | 2024 | 2024 |

**. Correlation is significant at the 0.01 level (2-tailed).

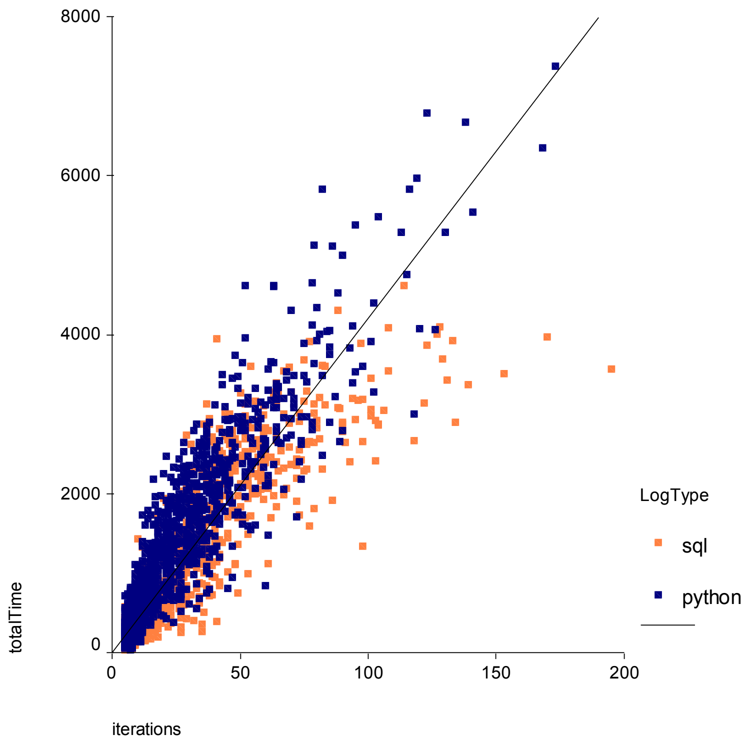


**Fig. 4.3:** Scatter plot of iterations and total time spent per session

The special shape of the scatter plot of *iterations* and *totalTime* suggests that students belong to different groups in terms of working speed (total time per iteration). The linear interpolation graph in Fig. 4.3 visually divides the population into two groups: one performing above average (*"slow"*), the other performing below average (*"fast"*).[589] Mathematically, different classes of working speed can be identified by logarithmized time-per-iteration ratios. Before applying such a classification, the

[589] In this case, being above the average refers to a slow working speed, since the corresponding ratio is time per iteration, i.e., those students spend more time per iteration than the average.

time-per-iteration ratio must be standardized in order to get a reference value. A feasible method to standardize the ratios and to categorize the population into classes is given by the following two equations (with *n* as the number of cases):

$$z_{language} = \frac{\frac{1}{n}\sum_{i=1}^{n} totalTime_i}{\frac{1}{n}\sum_{i=1}^{n} iterations_i} \qquad \textbf{Eq. 4.1}$$

$$class_i = -\left\lfloor \log_2\left(\frac{totalTime_i}{iterations_i} : z_{language}\right) + \frac{1}{2} \right\rfloor \qquad \textbf{Eq. 4.2}$$

The average time-iteration ratios ($z$) are 51.5 for Python and 39.3 for SQL (seconds per iteration). Thus, students need 51.5 seconds to work on one Python execution and 39.3 seconds to work on one SQL execution on average.

The difference between SQL and Python in terms of working speed is highly significant. Since the KOLMOGOROV-SMIRNOV test found that neither *totalTime* nor *iterations* is normally distributed, the non-parametric MANN–WHITNEY *U-test* was performed in place of the common *t-test*. The U-test confirmed *different* group means with a significance of $p<0.001$. Accordingly, students spent significantly more time for one Python code execution than for one SQL query execution.

In order to classify students, the individual time-per-iteration ratio of a student must be divided by the language-specific standard ratio $z$. The *logarithm dualis* (rounded to integer) assigns students to working speed classes. The logarithm dualis is chosen here, so that the class index can be interpreted. Accordingly, all cases within $class_0$ represent the average, while $class_1$ works at double speed, and $class_{-1}$ is half as fast as the average.

For example, a student executes 20 SQL queries in 5 minutes or 300 seconds, respectively. The individual time-per-iteration ratio of this student is 15.0 seconds per query. The result of dividing this ratio (15.0) by the language-specific ratio of SQL (39.3) and applying the logarithm dualis is approximately -0.88 which is then rounded down to -1.0. Finally, the sign is changed to +1.0. Therefore, this student belongs to $class_1$ and works at double speed compared to the average. The corresponding classification based on this approach is visualized in Fig. 4.4.

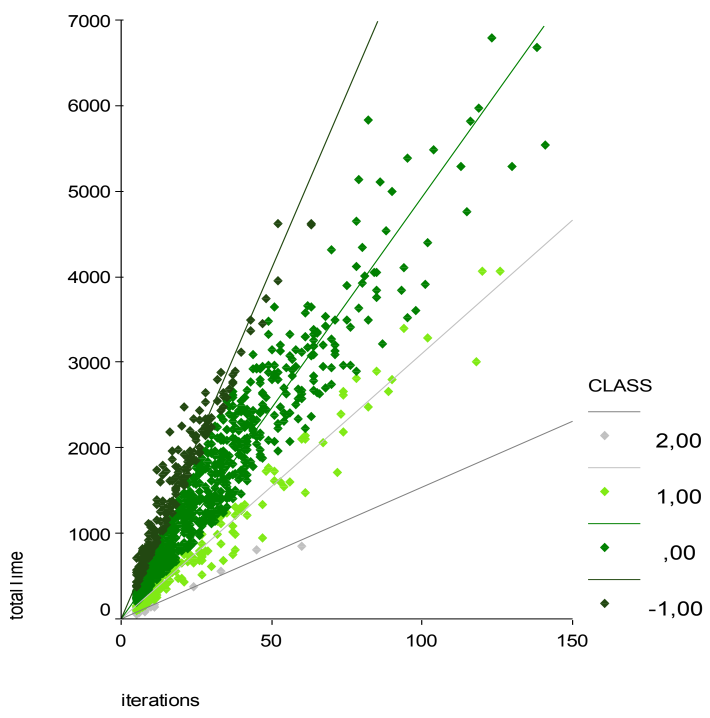


**Fig. 4.4:** Classes identified by the time-per-iteration ratio (Python only)

The scatter plot based on different classes visually confirms that students have different working speeds. Most students belong to *class*$_0$, which represents the average, i.e., 39.3 seconds to execute an SQL query. However, there are also students who need twice as much time (*class*$_{-1}$) while other students are working four times faster than the average (*class*$_2$).

Above all, the dataset and the corresponding statistical analysis clearly reflected some deficits of the summer semester data set. Based on a review of the technical environment, three major limitations could be identified:

1) All sessions were anonymous. It was impossible to observe a single student over time. Similarly, it was impossible to observe how a single student mastered SQL compared to Python and vice versa.

2) The particular learning objective of a single student was unknown. For an external observer, it was impossible to distinguish between aimless or purposive activities on the playgrounds.

3) Since students were not observed in a controlled environment, it was impossible to identify idle time.

In consequence, the exploratory analysis of the summer semester dataset only indicated different working speeds among the students. In view of the limitations, it was decided to prepare the playgrounds for the next semester in order to get a new dataset with more detailed information.

### 4.2.2 Winter Semester 2008/2009

Before the winter semester of 2008/09, the playgrounds were modified. Students now have to pass an authentication before using the playgrounds. This resolved the first limitation (anonymous sessions). The authentication is based on the chair's internal registration system for tutorials and courses.[590] Due to the authentication data, all activities on the playgrounds can be unambiguously assigned to individual students, who can also be assigned to particular lectures.[591] This was important in that particular winter semester, since two different lectures discussed SQL and promoted using the SQL playground for practicing.[592]

Besides these technical modifications, the SQL playground was extended by an interactive SQL trainer. A trainer-specific log file could fix the second limitation (unknown objective). The SQL Trainer provided 15 exercises with different levels of difficulty ranging from simple "SELECT-FROM-WHERE" queries to more difficult aggregation queries (see appendix A.1). Despite the different levels of difficulty, all students were expected to have the ability required for solving all exercises.

It has to be pointed out that due to didactical reasons the exercise order was not strictly in line with the supposed difficulty level of the exercises. The introductory page of the SQL trainer informed the students about this issue. In order to encourage the students to try all exercises, they should not assume that a higher exercise number necessarily reflects a higher level of difficulty. The SQL trainer allowed students to skip exercises, so they were not forced to solve the exercises in their order of appearance. In the following, the corresponding dataset is called *"Exercise Dataset"*.[593]

While working with the SQL Trainer, students can write and test SQL queries in the same way they do on the standard SQL playground (see Fig. 4.5). If students are un-

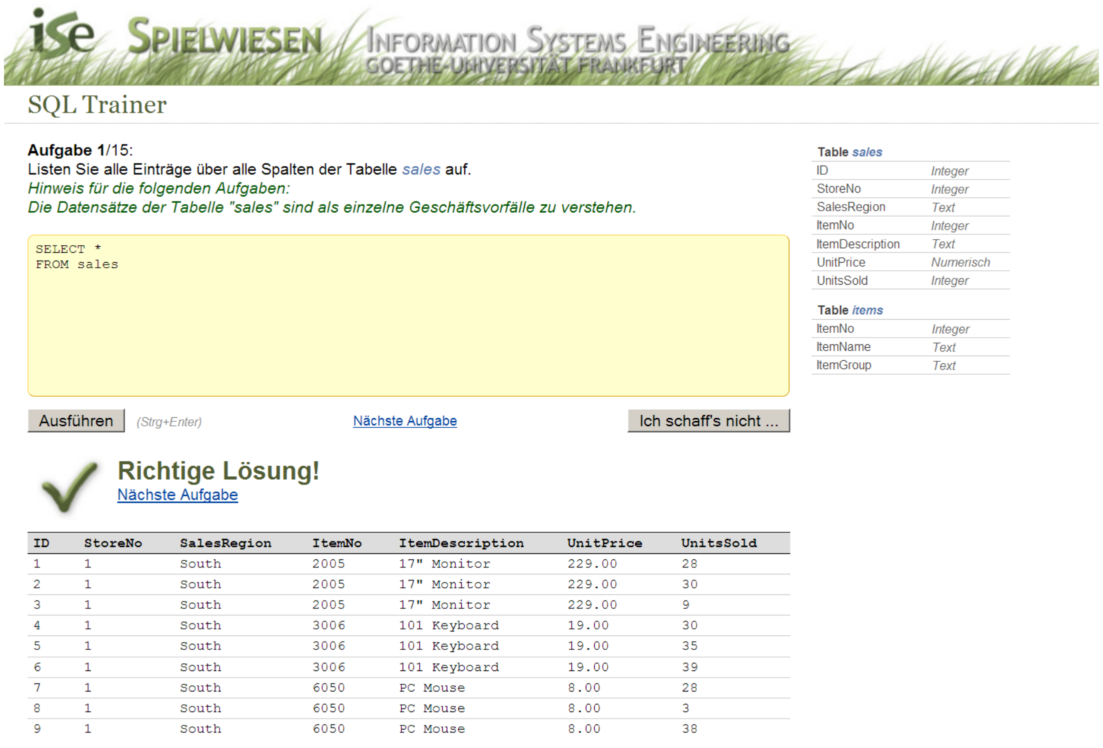
[590] BIGGELEBEN, HOLTEN (2009a).

[591] The identification of individuals is *not* based on personal data. Technically, the authentication mechanism returns a unique ID based on a consecutive number used by the registration system. This ID is used in the log files to distinguish between individuals. Accordingly, the datasets gained from the log files are anonymous. In addition, the playgrounds' login pages inform students that the activities on the playgrounds are logged, and that this data is subject to statistical analyses.

[592] These two lectures were *"Introduction to Information Systems"* (second semester bachelor students) and *"Management Information Systems"* (second semester master students).

[593] Another study using this dataset while focusing on the relationship of e-learning and learning success is given by BIGGELEBEN *et al.* (2009a).

able to solve an exercise, they can click on special link to see the sample solution for that particular exercise.



**Aufgabe 1**/15:
Listen Sie alle Einträge über alle Spalten der Tabelle *sales* auf.
*Hinweis für die folgenden Aufgaben:*
*Die Datensätze der Tabelle "sales" sind als einzelne Geschäftsvorfälle zu verstehen.*

```
SELECT *
FROM sales
```

| ID | StoreNo | SalesRegion | ItemNo | ItemDescription | UnitPrice | UnitsSold |
|---|---|---|---|---|---|---|
| 1 | 1 | South | 2005 | 17" Monitor | 229.00 | 28 |
| 2 | 1 | South | 2005 | 17" Monitor | 229.00 | 30 |
| 3 | 1 | South | 2005 | 17" Monitor | 229.00 | 9 |
| 4 | 1 | South | 3006 | 101 Keyboard | 19.00 | 30 |
| 5 | 1 | South | 3006 | 101 Keyboard | 19.00 | 35 |
| 6 | 1 | South | 3006 | 101 Keyboard | 19.00 | 39 |
| 7 | 1 | South | 6050 | PC Mouse | 8.00 | 28 |
| 8 | 1 | South | 6050 | PC Mouse | 8.00 | 3 |
| 9 | 1 | South | 6050 | PC Mouse | 8.00 | 38 |

| **Table** *sales* | |
|---|---|
| ID | *Integer* |
| StoreNo | *Integer* |
| SalesRegion | *Text* |
| ItemNo | *Integer* |
| ItemDescription | *Text* |
| UnitPrice | *Numerisch* |
| UnitsSold | *Integer* |

| **Table** *items* | |
|---|---|
| ItemNo | *Integer* |
| ItemName | *Text* |
| ItemGroup | *Text* |

**Fig. 4.5:** Web-based SQL Trainer

In order to solve an exercise, the database output and the SQL query must be in line with the sample solution. A sample solution consists of both the correct database output and the correct SQL query.[594] Correct queries, however, might differ in terms of spaces, line breaks, as well as the use of lower and upper case letters. Therefore, the correctness of the SQL query is not based on a strict string comparison but is determined by the LEVENSHTEIN distance between the sample solution and the query executed by the student.[595] For example, the following four SQL queries are correct solutions of exercise 13 (see Appendix A.1):

594 A sample solution might contain multiple correct database outputs as well as multiple correct SQL queries.

595 LEVENSHTEIN (1966). Before calculating the LEVENSHTEIN distance, all line-breaks and unnecessary spaces are removed from both the executed SQL query and the sample solution query. Moreover, both queries are transformed to lower case letters. Given that the database output perfectly matches (one of) the sample solution outputs, an SQL query is evaluated as correct if the LEVENSHTEIN distance between the sample solution query and the executed query measured in percent is less than 60%. This threshold was determined during the test phase of the SQL trainer.

```
/* 1) correct sample solution */
SELECT    ItemGroup
FROM      items
GROUP BY  ItemGroup

/* 2) also correct (different notation) */
select ItemGROUP
from items group by itemgroup

/* 3) alternative correct sample solution */
SELECT    DISTINCT ItemGroup
FROM      items

/* 4) also correct (different syntax) */
select distinct itemgroup from items
```

**Fig. 4.6:** Correct solutions of exercise 13

Finally, the third limitation (idle time) could be mitigated by implementing precise client-side timers that allow differentiating between activity and inactivity on the playgrounds. Additionally, students were encouraged to use the SQL trainer as a means of self-control with regard to the final exam in which queries must be solved correctly and within given time limits.

During the winter semester of 2008/09, all queries executed by students were recorded in a specific log file. Besides typical client information and the executed SQL query, this log file contains the authentication and precise timing data. The available variables are summarized in the following table Tab. 4.3.

**Tab. 4.3:** Data logged by the SQL Trainer

| Variable | Example | Description |
|---|---|---|
| timestamp | 2009-02-01 14:26:18 | Calendar time of query execution |
| client_ip | 84.176.14.20 | Client's IP address |
| session_id | F3C25C7…A1966F | 32 byte session identifier |
| user | 00001234 | Unique personal ID (not to be confused with the students' matriculation number) |
| status_code | OK | *"OK"* for syntactically correct queries, *"ERROR"* for syntactical errors, *"SOLVED"* for correct solutions, and *"CHEATED"* for checking the sample solution |
| database | SQL_trainer_1 | Database. *"SQL_trainer_1"* refers to the SQL Trainer |
| total_time | 3.750 | Total time between page load and query execution |
| focus_time | 3.750 | Less than total time if the browser window loses the focus[596] |
| busy_time | 3 | Only timed during keyboard or mouse activity |
| keystrokes | 10 | Number of pressed keys[597] |

[596] The variable "*focus_time*" represents the total time interval in which the browser window (or tab) was active. If a student minimizes the browser window or switches to another application, for example, to check his or her e-mails, the playground loses the focus. Accordingly, this timer can be used to identify idle time.

[597] The variable "*keystrokes*" is another candidate to identify idle time.

| Variable | Example | Description |
|---|---|---|
| exercise | 1 | Exercise number |
| query | SELECT * FROM… | Executed SQL query |

Each student has only one attempt per session to solve an exercise. An attempt, however, can consist of an unlimited number of query executions. If a student visits the trainer twice or more, only the first session will be considered. Likewise, if a student looks for the sample solution, the exercise cannot be solved any longer (internally marked as *"cheated"*; cp. *status_code* in Tab. 4.3). For example, a student uses the SQL Trainer for the first time, and solves the first five exercises. Assuming that the following exercises are too difficult for this student, he or she looks at the sample solutions of the remaining exercises. Later, the same student uses the SQL Trainer again, now being able to solve all 15 exercises. However, only the five exercises of the first session are rated as valid attempts.

Generally, working time is based on the variable *focus_time* (see Tab. 4.3). The dataset was inspected to find patterns and to develop an approach that allows identifying idle time, which should be excluded from statistical analyses. Therefore, if *focus_time* is less than 90 (1.5 minutes), no idle time is assumed, and *focus_time* is taken as the working time. Otherwise, the ratio of *focus_time* and *busy_time* is checked whether or not it appears plausible. According to the dataset inspection, a ratio greater than 3 appeared questionable. For those ratios, the working time is limited to three times the *busy_time* (which refers to actual keyboard and mouse activity). Alternative approaches using the variables *total_time, busy_time*, *focus_time, keystrokes*, or combinations of these were also tested. However, these tests showed that the presented approach can identify idle time more accurate than alternative approaches. Finally, if not otherwise noted, time measures are always given in seconds.

## 4.3 Data Exploration

The SQL Trainer was released on February 1st, 2009.[598] The final exam of the lecture of interest was on February 23rd, 2009. In total, 20,475 SQL queries were executed between these two dates.[599] The SQL trainer was used by 233 distinct individuals. After removing the log file entries of staff members, tutors, and students who did not attend the second semester lecture, the dataset contains sessions of 170 bachelor students. These students made over 2,000 solution attempts based on 8,756 considered

[598] The late release of the SQL Trainer at the end of the lecture was intended, since students were expected to start studying as late as possible. An early release was expected to produce a biased dataset with mainly invalid attempts.

[599] Exact time interval: 2009-02-01 00:00:00 – 2009-02-22 23:59:59.

query executions.[600] For comparison, over 500 students registered for the tutorials and 382 students took the final exam at the end of winter semester 2008/2009.

The subsequent data exploration pursues the following strategy:

- The first analyses focus on the solution capability, solution effort, and the individual working speed of students in order to gain descriptive statistics and to develop a basic understanding of the dataset.
- Since the population is heterogeneous with regard to solution capability, students are classified and assigned to identical groups of solution capability.
- An approach based on *Simple Random Subsamples* and *Bootstrapping* is introduced. This approach allows deriving virtual development teams from the empirical data.
- Based on these virtual development teams, statistical analyses focus on the central question of this exploratory study, that is, to what extent it is possible to estimate effort when actual effort is unaffected by estimates. Different estimation approaches are evaluated to develop an understanding of estimation accuracy as well as the relationship between estimates and actual development effort.
- During this evaluation the impact of effects like PARKINSON and *Procrastination* as well as simple forms of *project control* are discussed and analyzed. Finally, the consideration of individual differences is explored.

### 4.3.1 Analysis of Solution Capability

As a first step, the cases of the Exercise Dataset were aggregated in order to get solution frequencies, general solution capability, as well as average solution effort. Tab. 4.4 summarizes these descriptive statistics. In this table, the column *"AttemptedPct"* shows how many students have tried to solve a particular exercise. The column *"SolvedPct"* gives the corresponding rate of success. Both figures are given as percentages of the total sample size, i.e., 170 bachelor students. The supposed level of difficulty of the exercises is given by an ordinal scale, ranging from *A* to *E* with *A*

[600] As mentioned before, only the first session on the SQL trainer was considered.

representing easy exercises and *E* very difficult exercises. The effort-oriented figures consider successful solutions only. It is intended that attempts, which did not lead to a solution or in which students cheated do not influence the time measures.

**Tab. 4.4:** Solution attempts grouped and ordered by the original exercise numbers[601]

| **Original exercise number** | **Supposed level of difficulty** | **Altered exercise number** | **Min. solution time [sec]** | **Mean solution time [sec]** | **Max. solution time [sec]** | **Standard deviation** | **Attempt-edPct** | **Solved-Pct** |
|---|---|---|---|---|---|---|---|---|
| 1 | A | 2 | 12 | 64 | 1,168 | 56.4 | 96% | 84% |
| 2 | A | 1 | 17 | 71 | 347 | 111.4 | 97% | 84% |
| 3 | B | 4 | 13 | 154 | 975 | 41.0 | 94% | 66% |
| 4 | C | 9 | 24 | 275 | 1,295 | 205.3 | 94% | 24% |
| 5 | B | 5 | 25 | 120 | 450 | 82.4 | 92% | 52% |
| 6 | B | 3 | 24 | 77 | 292 | 36.4 | 92% | 78% |
| 7 | B | 7 | 34 | 194 | 905 | 176.5 | 88% | 41% |
| 8 | D | 13 | 138 | 529 | 1,651 | 96.3 | 86% | 13% |
| 9 | C | 10 | 51 | 226 | 757 | 320.6 | 81% | 23% |
| 10 | C | 12 | 29 | 196 | 661 | 176.9 | 63% | 14% |
| 11 | C | 8 | 44 | 131 | 623 | 182.4 | 74% | 31% |
| 12 | C | 11 | 95 | 245 | 1,177 | 151.3 | 69% | 23% |
| 13 | B | 6 | 20 | 65 | 227 | 386.1 | 69% | 51% |
| 14 | E | 15 | 297 | 397 | 473 | 169.0 | 57% | 2% |
| 15 | E | 14 | 82 | 218 | 654 | 74.2 | 47% | 6% |

This aggregated view of the data reveals that the first five exercises were addressed by more than 90% of the students.[602] The number of attempts, however, is continuously decreasing over the 15 exercises. The last exercise was addressed by only 47% of all students. This shows that lots of students refrained from solving *all* exercises. Besides disinterest or other external influences, students might have given up, because some exercises were too difficult, or some students might have set a time limit for the session. Nevertheless, the aggregated view exposes that the general solution capability is significantly different across the students. Especially, the exercises 8, 10, 14, and 15 appear to be exceptionally difficult. The following Fig. 2.1 visualizes the percentages of solution attempts and correct solutions.

[601] The exercise number will be altered to match the order of solution capability in a later step.

[602] The fact that the first exercise is not attempted by 100% of the students is based on the technical possibility to skip exercises.

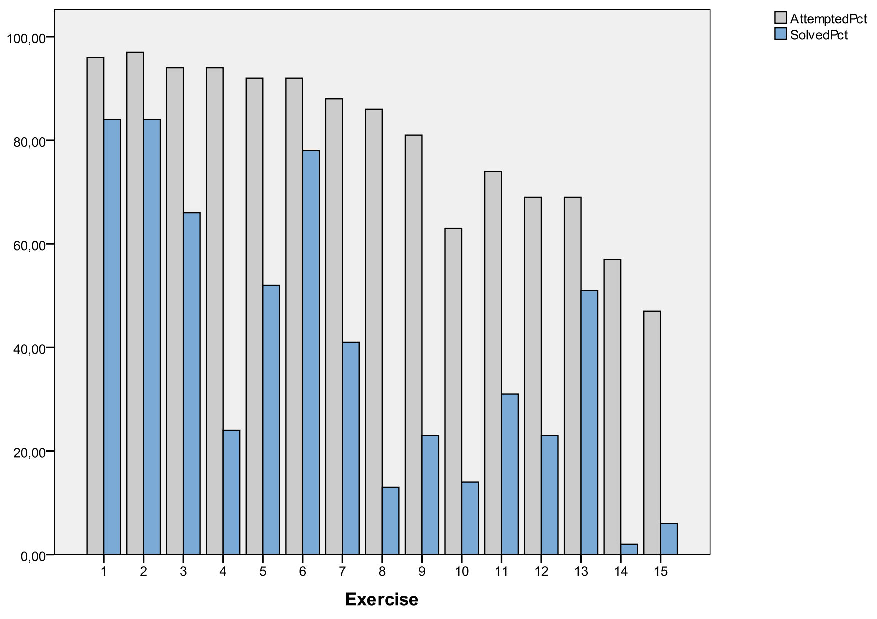


**Fig. 4.7:** Solution attempts and correct solutions in original order

The actual solution capability of students (see Fig. 4.7; *SolvedPct*) matches the supposed difficulty levels of the exercises. Exercise 8 is not difficult in terms of SQL, but its correct solution requires a proper discount of *value-added tax*. Exercise 14 and 15 address *joining* two tables, which was only shortly introduced in the lecture notes. Exercise 10, however, had an unexpected low solution percentage. The solution of this exercise requires the correct use of the aggregation function *COUNT* without a *GROUP BY* clause. After the SQL Trainer was released, it turned out that the lecture notes did not discuss this type of SQL queries. This lack of information is supposed to be responsible for the low solution percentage.

Before turning to further analyses, the exercise numbers are altered so that they reflect the actual solution capability. This alteration is beneficial for the readability of the remainder of this chapter, as it leads to a strong correlation of exercise number and solution capability. Accordingly, a higher exercise number refers to a higher empirical level of difficulty as well as a lower solution percentage.

After reordering the exercises, the *mean* solution time now strongly correlates with the supposed level of difficulty ($\rho=0.850$; $p<0.01$; see Tab. 4.5).[603] The *minimal* solution time also shows a strong correlation with difficulty ($\rho=0.876$; $p<0.01$).

[603] In this case, the exercise number, typically a *nominal* scaled variable, can be understood as an *ordinal* scaled variable as it reflects the actual level of difficulty. In consequence, SPEARMAN'S rank correlation coefficient is used to determine the correlation between exercise and the solution time.

**Tab. 4.5:** Correlation of difficulty and solution time

| | | | Exercise | Mean | Min | Max |
|---|---|---|---|---|---|---|
| Spearman's rho | Exercise | Correlation Coefficient | 1,000 | ,850** | ,876** | ,300 |
| | | Sig. (2-tailed) | . | ,000 | ,000 | ,277 |
| | | N | 15 | 15 | 15 | 15 |
| | Mean | Correlation Coefficient | ,850** | 1,000 | ,792** | ,564* |
| | | Sig. (2-tailed) | ,000 | . | ,000 | ,028 |
| | | N | 15 | 15 | 15 | 15 |
| | Min | Correlation Coefficient | ,876** | ,792** | 1,000 | ,193 |
| | | Sig. (2-tailed) | ,000 | ,000 | . | ,491 |
| | | N | 15 | 15 | 15 | 15 |
| | Max | Correlation Coefficient | ,300 | ,564* | ,193 | 1,000 |
| | | Sig. (2-tailed) | ,277 | ,028 | ,491 | . |
| | | N | 15 | 15 | 15 | 15 |

**. Correlation is significant at the 0.01 level (2-tailed).

*. Correlation is significant at the 0.05 level (2-tailed).

The alteration of the original exercise numbers, based on the order of appearance in the SQL trainer, to exercise numbers that match the solution capability is given in Tab. 4.4 as well as appendix A.1. The new order of exercises and the corresponding percentages of attempts and correct solutions are visualized in the following Fig. 4.8.

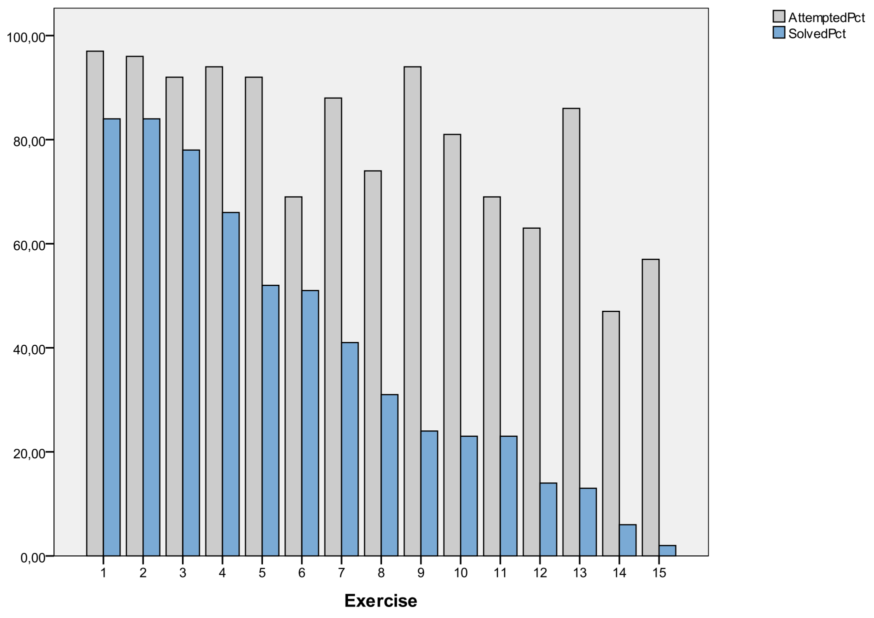


**Fig. 4.8:** Solution attempts and correct solutions in altered order

To prevent ambiguity, the data given in Tab. 4.4 is also presented in the altered order of solution ability (see Tab. 4.6). In the following, exercise numbers refer to this altered order.

**Tab. 4.6:** Solution attempts grouped and ordered by the altered exercise numbers

| Altered exercise number | Supposed level of difficulty | Original exercise number | Min. solution time [sec] | Mean solution time [sec] | Max. solution time [sec] | Standard deviation | Attempt-edPct | Solved-Pct |
|---|---|---|---|---|---|---|---|---|
| 1 | A | 2 | 17 | 71 | 347 | 111.4 | 97% | 84% |
| 2 | A | 1 | 12 | 64 | 1,168 | 56.4 | 96% | 84% |
| 3 | B | 6 | 24 | 77 | 292 | 36.4 | 92% | 78% |
| 4 | B | 3 | 13 | 154 | 975 | 41.0 | 94% | 66% |
| 5 | B | 5 | 25 | 120 | 450 | 82.4 | 92% | 52% |
| 6 | B | 13 | 20 | 65 | 227 | 386.1 | 69% | 51% |
| 7 | B | 7 | 34 | 194 | 905 | 176.5 | 88% | 41% |
| 8 | C | 11 | 44 | 131 | 623 | 182.4 | 74% | 31% |
| 9 | C | 4 | 24 | 275 | 1,295 | 205.3 | 94% | 24% |
| 10 | C | 9 | 51 | 226 | 757 | 320.6 | 81% | 23% |
| 11 | C | 12 | 95 | 245 | 1,177 | 151.3 | 69% | 23% |
| 12 | C | 10 | 29 | 196 | 661 | 176.9 | 63% | 14% |
| 13 | D | 8 | 138 | 529 | 1,651 | 96.3 | 86% | 13% |
| 14 | E | 15 | 82 | 218 | 654 | 74.2 | 47% | 6% |
| 15 | E | 14 | 297 | 397 | 473 | 169.0 | 57% | 2% |

### 4.3.2 Analysis of Solution Effort

The effort-oriented figures in the preceding Tab. 4.6 (*Min/Mean/Max*) show extreme differences in the required effort for exercise solution. The highest effort ratio between the fastest and slowest solution can be found for exercise 2, which is 1:97 (Min/Max). The fastest students spent 12 seconds for the solution, while the slowest student spent nearly 20 minutes. The min-max-ratios of the other exercises usually range from 1:12 to 1:26. For all extreme values, the cases were manually inspected in order to verify that the figures do not result from measurement errors.[604]

To demonstrate the extreme but plausible figures of exercise 2 (1,167.5 seconds/19.5 minutes), the complete attempt of that particular student is listed in Appendix A.2.[605] Interestingly, the student did not get discouraged by that time-consuming attempt and continued to try all 15 exercises. In total, this particular student successfully solved 7 exercises, which suggests a noteworthy ambition.

Such extreme cases might have a considerable impact on the exploration and further statistical analyses. Accordingly, those cases must be checked whether they represent

[604] Extremely fast solution were inspected to make sure that the number of keystrokes was in line with the length of the executed query and to exclude that those students copied and pasted prepared solutions. Extremely slow solutions were inspected to exclude that they were based on idle time or nonsensical query executions.

[605] The average time-per-iteration ratio of the SQL trainer is 37.4 seconds per SQL query. The aforementioned student has an individual time-per-iteration ratio of 50.53. However, this student would still be classified as average ($class_0$) in terms of working speed (cp. Section 4.2.1).

*outliers*. Extreme values, however, are not outliers in general. Following BARNETT and LEWIS:

> *"From the earliest times, there has been a concern for 'unrepresentative', 'rogue', or 'outlying' observations in sets of data. There are often seen as contaminating the data: reducing and distorting the information about the data source or generating mechanism. It is natural to seek means of interpreting or categorizing outliers and methods for handling them – sometimes perhaps rejecting them to restore the propriety of the data, or at least adopting methods of reducing their impact in any statistical analysis. [... However,] outlying observations do not inevitably 'perplex' or 'mislead'; they are not necessarily 'bad' or 'erroneous' [...]"*[606]

Additionally, BARNETT and LEWIS define:

> *"An outlier in a set of data [is] an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data. The phrase 'appears to be inconsistent' is crucial. It is a matter of subjective judgment on the part of the observer whether or not some observation (or set of observations) is picked out for scrutiny."*[607]

For that reason, two simple methods that help identifying outliers were applied. As a preparing step, the Exercise Dataset was statistically analyzed in order to get the mean $\mu_i$ , the standard deviation $\sigma_i$, as well as the lower and upper quartiles $Q1_i$ and $Q3_i$ for each exercise $i$. The first method marks cases as outliers if they a $k$ standard deviations away from the mean.[608] Accordingly, extreme values that are outside the following range represent potential outliers:

$$\left[\mu_i - k\sigma_i, \mu_i + k\sigma_i\right] \quad with\, k > 0 \qquad \textbf{Eq. 4.3}$$

For $k=2$, 52 cases (5.2%) and, for $k=3$, 24 cases (2.4%) were identified as potential outliers. Generally, only slow solutions were affected. The manual inspection of the dataset, however, showed that there is not necessarily a notable gap between the mass of the data and the potential outliers. As an example, Fig. 4.9 shows the distribution of solution time concerning exercise 10. For $k=2$, only the highest solution time (757 sec) was identified as an outlier. In contrast, the slightly isolated cases between 400 and 650 seconds were not marked as outliers. Generally, it is questionable

[606] BARNETT (1994), p. 3.
[607] BARNETT (1994), p. 7.
[608] E.g., LEONG, AUSTIN (2005).

whether such extreme values are inconsistent with the rest of the dataset, or whether they are special but consistent cases of solution effort.

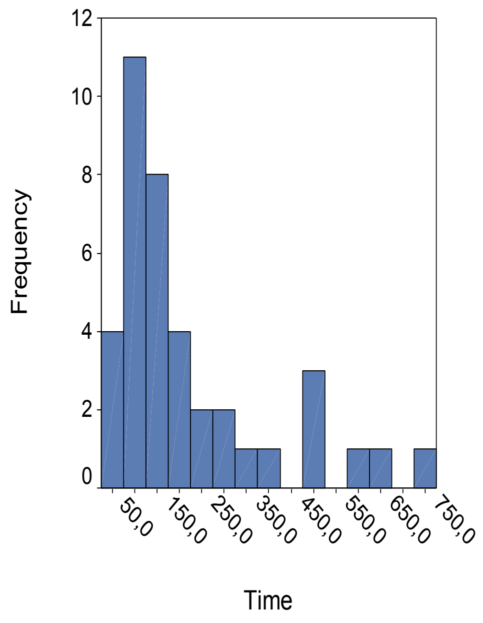


**Fig. 4.9:** Exemplary distribution of solution time of exercise 10

The second method uses the *interquartile range ($Q3_i$-$Q1_i$)* to identify outliers.[609] Values that are not inside this range are marked as potential outliers:

$$\left[Q1_i - k(Q3_i - Q1_i), Q3_i + k(Q3_i - Q1_i)\right] \quad with\ k > 0 \qquad \textbf{Eq. 4.4}$$

For *k=2*, 63 cases (6.3%) and, for *k=3*, 40 cases (4.0%) were identified as potential outliers. Like the first approach, only slow solutions were affected, and this method also put the cutoff point for high values inside the mass of the data.

To conclude, the dataset does not contain evident outliers that can clearly be classified as *inconsistent* with the remainder of the dataset. It is supposed that extremely slow solution belong to the dataset just like extremely fast solutions. In reference to real-world software development, it is also supposed that developers might get stuck

[609] BARNETT (1994); LEONG, AUSTIN (2005).

during a solution and need a large amount of effort to complete a work package.[610] In consequence, no cases were removed from the Exercise Dataset.

### 4.3.3 Analysis of Working Speed

The preceding analysis based on the summer semester dataset indicated that students have different working speeds (see Section 4.2.1). Moreover, the dataset suggested that students also have an individual working speed that is independent of the level of difficulty and the programming language.[611]

If this is true, the comparison of the individual working speeds measured on the normal SQL playground and the Python Playground must show a positive correlation. Since students do not explicitly solve exercises on these playgrounds, we can only observe the ratio of code execution per time. However, if students have an individual working speed, this will have a measureable effect on this ratio. Accordingly, the *null hypothesis* of this test is that the corresponding correlation coefficient is zero.

The following analysis is based on 50,475 SQL queries and Python program executions. This dataset does not contain code *repetitions* or code *examples* that are provided by the playgrounds. The cases are aggregated per student in order to get the total working time and code executions for both SQL and Python. Aggregated cases that do not contain values of both playgrounds cannot be compared, and, thus, these cases are not considered. In total, 273 students have worked on *both* playgrounds. These 273 cases represent the relevant dataset for the correlation analysis. On average, the students have executed 59 SQL queries and 116 Python programs during the winter semester of 2008/09. The result of the correlation analysis is given in Tab. 4.7.

[610] The highest deviation of expected and actual development effort experienced by the author was 16 hours instead of 10 minutes, although the task was simple. The particular solution should generate and send appointments via e-mail so that typical PIM systems like Outlook Exchange present a typical dialog asking whether to accept or decline the appointment. Such e-mails must have an additional part that contains the appointment data in the *iCalendar* format. Timestamps must be given in the *Coordinated Universal Time* (UTC) format, for example, "20090101T120000Z". Due to a tiny flaw in the code, the hours of the timestamp did not have a leading zero for one-digit values. Therefore, this flaw is not detected for two-digit hours. While the author tested the solution with two-digit hours, another test user worked with one-digit hours internally (8:00:00), which was not evident, because the visible test time (10:00:00) had two digits as it was given in *Central European Time* (CET). In consequence, the error only appeared on the second test system while the first test system worked perfectly. Since the second test system used a different Outlook version, a different operating system, and a different e-mail provider, the error was search anywhere else but in the routine responsible for the timestamp generation. Finally, the error was detected after two days of troubleshooting.

[611] The weak correlations ($r<0.2$) of time (*totalTime*) and code level (*avgCL*), code length (*avgL*), and Lines of Code (*avgLOC*) suggest an individual working speed of students (see Section 4.2.1).

**Tab. 4.7:** Correlation of SQL- and Python-related working speeds

| | | SQL Working Speed | Python Working Speed |
|---|---|---|---|
| SQL Working Speed | Pearson Correlation | 1 | ,546** |
| | Sig. (2-tailed) | | ,000 |
| | N | 273 | 273 |
| Python Working Speed | Pearson Correlation | ,546** | 1 |
| | Sig. (2-tailed) | ,000 | |
| | N | 273 | 273 |

**. Correlation is significant at the 0.01 level (2-tailed).

If all cases are considered, the correlation coefficient is $\rho=0.546$ ($p<0.01$). Therefore, the null hypothesis must be rejected. This supports the presumption that students have an individual working speed, no matter whether they focus on SQL or Python. Additionally, if the sample is reduced to cases in which students executed a high amount of codes, e.g., more than 75 SQL queries or Python codes, the analysis will show a strong positive correlation of $\rho=0.736$ ($p<0.01$). Accordingly, students have an independent working speed that is independent of both the level of difficulty and the programming language (SQL and Python).

**Tab. 4.8:** Correlation of SQL- and Python-related working speeds (subsample)

| | | SQL Working Speed | Python Working Speed |
|---|---|---|---|
| SQL Working Speed | Pearson Correlation | 1 | ,736** |
| | Sig. (2-tailed) | | ,000 |
| | N | 56 | 56 |
| Python Working Speed | Pearson Correlation | ,736** | 1 |
| | Sig. (2-tailed) | ,000 | |
| | N | 56 | 56 |

**. Correlation is significant at the 0.01 level (2-tailed).

Finally, in order to get an overview of the distribution of working speeds given in the *Exercise Dataset*, the classification approach as introduced in Section 4.2.1 is applied. The average time-per-iteration ratio of successful exercise solutions is 37.7 seconds per SQL query. The following Tab. 4.9 presents the corresponding results.[612] Additionally, a box-plot of this classification is given in Fig. 4.10.

[612] This classification uses half classes. The working speed in percent is based on the average class 0. Therefore, class 1.0 (double speed), for example, represents a working speed of 200%. The percentages of half classes are based on multiples of two as well. Accordingly, class 1.5 refers to a working speed of 2 to the power of 1.5 which is approx. 2.83.

**Tab. 4.9:** Classification of working speed

| Class | Number of students | Average working speed in percent | Average time per query [sec] |
|---|---|---|---|
| -1.5 | 6 | 35% | 106.6 |
| -1.0 | 18 | 50% | 75.4 |
| -0.5 | 45 | 71% | 53.3 |
| ±0.0 | 57 | 100% | 37.7 |
| +0.5 | 24 | 141% | 26.7 |
| +1.0 | 10 | 200% | 18.8 |
| +1.5 | 1 | 283% | 13.3 |
| +2.0 | 1 | 400% | 9.4 |

The given classification was statistically tested by the non-parametric KRUSKAL-WALLIS *H-test*, an extension of the MANN–WHITNEY *U-test* that works for more than two groups: *"The MANN-WHITNEY U-test [is] the non-parametric analog to the t-test on means for independent groups [while] the KRUSKAL-WALLIS 'Analysis of Variance' [is] the non-parametric analog of the one-way ANOVA."* [613] The *H-test* confirmed that the classes are significantly different in terms of working speed (p<0.001; 1004 cases). The same result could be obtained when testing all cases including unsolved and cheated attempts (p<0.001; 1984 cases).

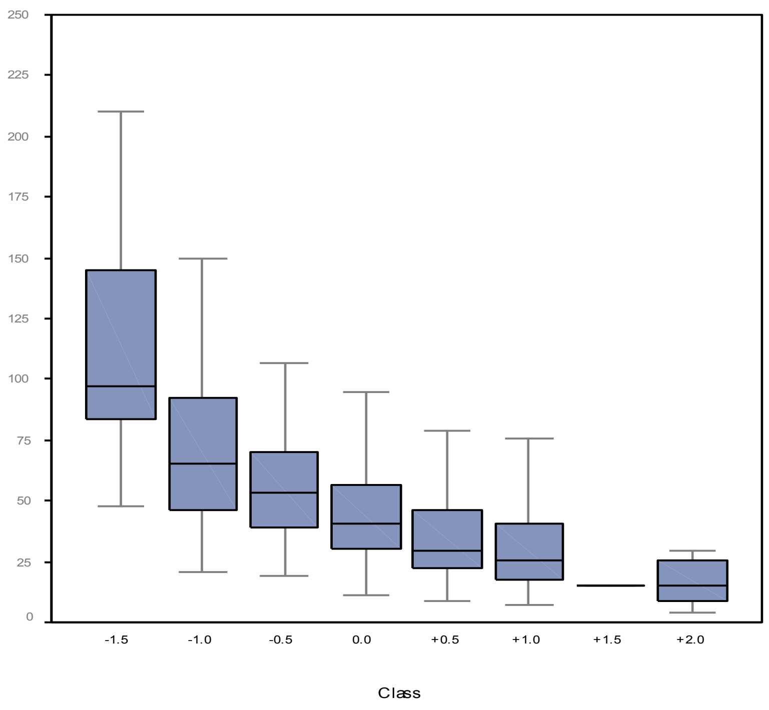


**Fig. 4.10:** Box-plot of identified working speed classes

[613] LEONG, AUSTIN (2005), p. 505.

### 4.3.4 Simple Random Sampling

So far, the exploratory analysis of the Exercise dataset showed that solution capability, solution effort, and working speed differ significantly among students. Therefore, this subsection starts to put the focus on the central question of this exploratory study, which is examining, whether it is possible to accurately estimate programming effort of individuals.

As presented in Section 2.3, a common technique to estimate effort is producing estimates by *analogy*. This approach requires either personal experiences or documented historical data of completed projects. Basically, the work packages of a new project are compared to this knowledge base, in order to find similar work packages of past projects.

In this exploratory study, the empirical dataset of the SQL Trainer is used as such a historical knowledge base. Since all students had to solve the same exercises, the exercise number leads to perfect analogy. While in real-world development projects work packages can usually be identified as only similar or comparable to previous work, the exercises solved by the students are not similar but *identical*. Additionally, all solution attempts are precisely documented in the Experience Dataset, and, accordingly, it is uncomplicated to determine the average effort per exercise as the foundation for analogy-based effort estimation (see Tab. 4.6).

The central question of this exploratory study is addressed by using *Simple Random Samples*, in order to simulate small developer teams as well as to estimate and analyze their development efforts.[614] *"A simple random sample is a sample in which every member of the population has the equal change of being chosen."*[615] In this study, a simple random sample is drawn by selecting *three* unique students who form a virtual developer team *(sampling without replacement*).[616] Each team member has to solve *five* arbitrary exercises as work packages. Since we only have 15 different exercises in total, a single exercise may be selected multiple times *(sampling with replacement*).[617] The estimated effort of a single team can be determined by summing up the mean efforts of each selected exercise. Moreover, the actual effort per student and exercise can also be looked up in the database. As a result, it is possible to determine both the actual effort and the expected effort for each random sample.

[614] BREAKWELL *et al.* (2006); DONNELLY (2004).
[615] DONNELLY (2004), p. 158.
[616] BREAKWELL *et al.* (2006).
[617] BREAKWELL *et al.* (2006).

#### 4.3.4.1 Groups of Solution Capability

Before the random samples can be drawn, the problem of cheated, unsolved, and unaddressed exercises must be fixed. Besides correct solutions (1,004 cases), the dataset contains lots of cheated (981 cases) and a few unsolved solutions (46 cases). These cases are supposed to have biased or distorted time measures, as they do not refer to successful exercise solutions. Furthermore, the dataset does not cover all student-exercise combinations, since it contains only 2,031 cases, while 2,550 cases (170 students multiplied by 15 exercises) are possible. There is a lack of information of 519 combinations of student and exercise number. Accordingly, in order to make all simple random samples comparable, only successful exercise solution should be considered.

***Determining Solution Capability***

For that reason, the students are grouped in terms of solution capability. In this case, solution capability is determined by the following simple approach:

1) The dataset is ordered by student and exercise number.

2) For each student, it is determined how many exercises were successfully solved in a row starting from the first exercise. The maximum exercise number is taken as the solution capability, and used to assign students to groups.

For example, a student who correctly solved exercise one to eight without cheating has a solution capability of eight. If the same student skipped exercise 5, his or her solution capability would be four. Based on this approach, students are assigned to groups of identical solution capability. It has to be pointed out that students of group $i$ are assigned to all lower groups $j$ with $j<i$ as well. Accordingly, the 1st group contains *all* students who were assigned to a group. In total, 151 out of 170 students were assigned to groups of solution capability. The remaining 19 students did not solve any exercise or skipped the first exercise.

For each group, simple random subsamples are drawn. At first, three students with the same solution capability $i$ are randomly selected. Afterwards, 15 exercises are randomly selected and assigned to the team members, each getting five exercises. During this assignment, all selected exercise numbers must be equal to or less than the group number $i$. For higher exercises numbers it cannot be guaranteed that the dataset contains useful information for that particular combination of student and exercise number. This approach ensures that all exercises can be solved by the randomly arranged teams. The simple random samples could be drawn for 13 groups, because less than 3 students reached a solution capability of 14 or 15, respectively.

A brief example helps understanding the process of how a random subsample is drawn. Group 10 contains 11 students. From this group, three students (A, B, and C) are randomly selected as developers who form a virtual development team. Next, five of the first 10 exercises are selected and assigned to one student. This is done three times so that each student has assigned five work packages (see Fig. 4.11).[618]

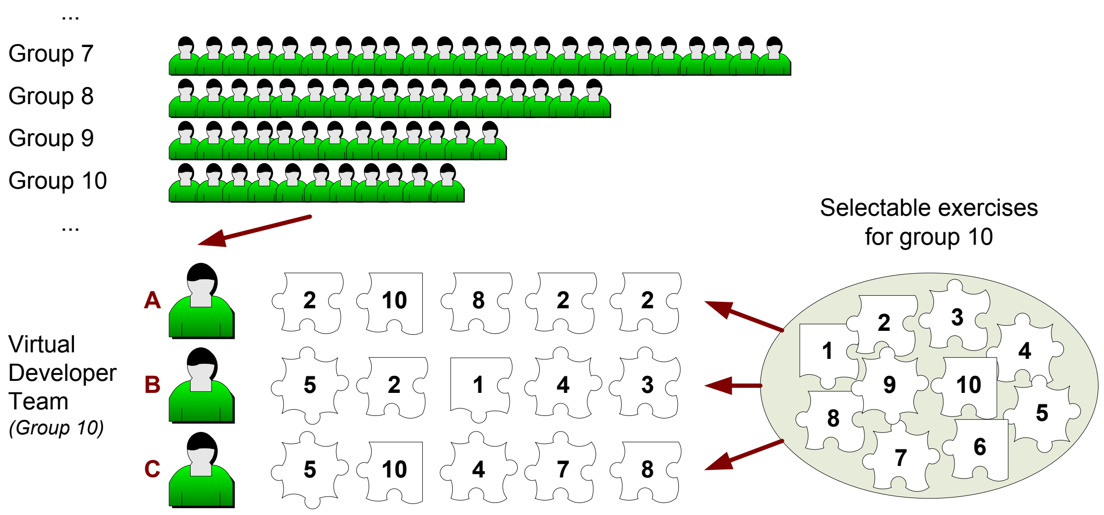


**Fig. 4.11:** Drawing simple random subsamples

The individual actual efforts per exercise as well as the average effort per exercise (see Tab. 4.6) are known. The following Tab. 4.10 describes an exemplary subsample, which shows that exercises can appear more than once. In this example, the expected effort, estimated by analogy, is 1861.1 (seconds) for all 15 work packages. The actual effort, however, is 2408.5 (seconds). This implies an effort overrun of 29.41%, which is in line with the commonly reported effort overruns of software development projects (see Section 3.1.3.1).

618 The list of work packages is shuffled after the selection process. Thus, developer C, for example, might start with two work packages, followed by developer B with one work package etc. The order of work has an effect on actual work time when considering forms of project control. This aspect is discussed later in this chapter.

**Tab. 4.10:** Exemplary random subsample

| Developer A | | | Developer B | | | Developer C | | |
|---|---|---|---|---|---|---|---|---|
| Exercise Number | Actual Effort [sec] | Mean effort [sec] | Exercise Number | Actual Effort [sec] | Mean effort [sec] | Exercise Number | Actual Effort [sec] | Mean effort [sec] |
| 2 | 42.500 | 64.1 | 5 | 69.000 | 119.9 | 5 | 118.750 | 119.9 |
| 10 | 319.750 | 225.7 | 2 | 33.750 | 64.1 | 10 | 225.000 | 225.7 |
| 8 | 295.000 | 131.2 | 1 | 26.000 | 71.4 | 4 | 678.750 | 154.3 |
| 2 | 42.500 | 64.1 | 4 | 29.500 | 154.3 | 7 | 206.000 | 193.9 |
| 2 | 42.500 | 64.1 | 3 | 59.250 | 77.2 | 8 | 220.250 | 131.2 |

| Work Packages | Actual Effort [sec] | Expected Effort [sec] | Effort Overrun [sec] | Effort Overrun % |
|---|---|---|---|---|
| 15 | 2408.5 | 1861.1 | 547.4 | 29.41% |

For each of the 13 groups of solution capability, 10,000 simple random samples are generated. This follows the basic idea of *Bootstrapping*, which allows statistical inference based on a small subsample of a population.[619] In total, 130,000 cases are available for the analysis of actual and estimated effort as well as the evaluation of overall estimation accuracy. The corresponding dataset is called *Bootstrap Dataset.* In order to ensure comparability, this dataset is generated only once, and all cases are stored in a database (1.95 million rows).[620]

### *Different Estimation Kernels*

In the remainder of this chapter, different *estimation kernels* will be discussed. An estimation kernel determines how estimated and actual efforts are processed. The estimated effort of a work package is generally based on the empirical average effort of the corresponding exercise.[621] Estimation kernels can modify the actual or estimated efforts in order to emulate certain effects that are supposed to be inherent in software development projects.

To recapitulate, the approach of simple random subsamples utilized in the remainder of this chapter is generally based on the following steps:

1) Students are grouped by solution capability.

2) For each group, 10,000 simple random subsamples are drawn.

[619] EFRON (1979a); EFRON (1979b); EFRON, TIBSHIRANI (1993).

[620] 130,000 (cases/development teams) multiplied by 15 (work packages/exercises).

[621] The average effort per exercise is calculated by considering the actual efforts of successful exercise solutions only. The average is not calculated per group but for the entire population (all students in the dataset). Cheated or unsolved exercises do not influence the averages.

3) A subsample represents a virtual development team. Each team consists of three developers or students, respectively, who are randomly picked out of the corresponding group of solution capability *(sampling without replacement*).

4) Each team must solve 15 exercises, which are randomly picked out of the pool of available exercises *(sampling with replacement*). The steps 1) to 4) generate the Bootstrap Dataset.

5) The actual and estimated efforts per team are processed and analyzed afterwards. In the following, analyses differ in the applied estimation kernel.

***Basic Estimation Kernel***

For the *basic* estimation kernel, the actual efforts are based on the unadjusted empirical solution effort of the corresponding exercise and student. Tab. 4.11 presents the results of this kernel. This table shows the effort deviation for each group of solution capability based on 10,000 iterations per group. The first group, for example, contains all 151 students that solved exercise one. On average, the actual effort of the virtual development teams based on the *first* group deviates from the estimates by only +0.2% (this value must converge towards zero). The fastest development team had an effort underrun of 73.5% in contrast to the slowest team that overran estimated effort by 360.8%.

**Tab. 4.11:** Overview of the Bootstrap Dataset (basic kernel)

| Group | Cases | Group members | Min. effort deviation [%] | Mean effort deviation [%] | Max effort deviation [%] |
|---:|---:|---:|---:|---:|---:|
| 1 | 10,000 | 151 | -73.5% | 0.2% | 360.8% |
| 2 | 10,000 | 132 | -77.8% | 1.0% | 1616.3% |
| 3 | 10,000 | 112 | -67.0% | 1.4% | 788.8% |
| 4 | 10,000 | 95 | -78.5% | 3.5% | 500.4% |
| 5 | 10,000 | 72 | -73.4% | 5.8% | 604.9% |
| 6 | 10,000 | 47 | -69.7% | 5.5% | 349.7% |
| 7 | 10,000 | 33 | -70.4% | 12.3% | 253.1% |
| 8 | 10,000 | 22 | -68.6% | 13.8% | 202.2% |
| 9 | 10,000 | 13 | -73.2% | 7.2% | 195.0% |
| 10 | 10,000 | 11 | -70.3% | 17.5% | 180.1% |
| 11 | 10,000 | 9 | -70.4% | 11.1% | 188.0% |
| 12 | 10,000 | 8 | -71.9% | 13.5% | 197.2% |
| 13 | 10,000 | 6 | -66.3% | 17.6% | 186.8% |

#### 4.3.4.2 Paradoxical Averages

Since the basic estimation kernel is based on average solution efforts, we expect that the actual and expected mean efforts must be consistent for all groups. Since higher group ranks refer to higher solution capability, we might even expect that higher ranked groups require less effort than expected. In fact, *the opposite is true*.

Paradoxically, the actual effort of each group is higher than the expected effort (see Tab. 4.11 and Fig. 4.12; with the exception of exercise one). For example, the mean effort deviation of the 10th group is 17.5% (see Tab. 4.11 and Fig. 4.12), *i.e.*, the actual efforts based on 10,000 iterations exceed the estimates by 17.5% (not approx. 0%) on average. These mean deviations reflect a systematic estimation error in this approach.

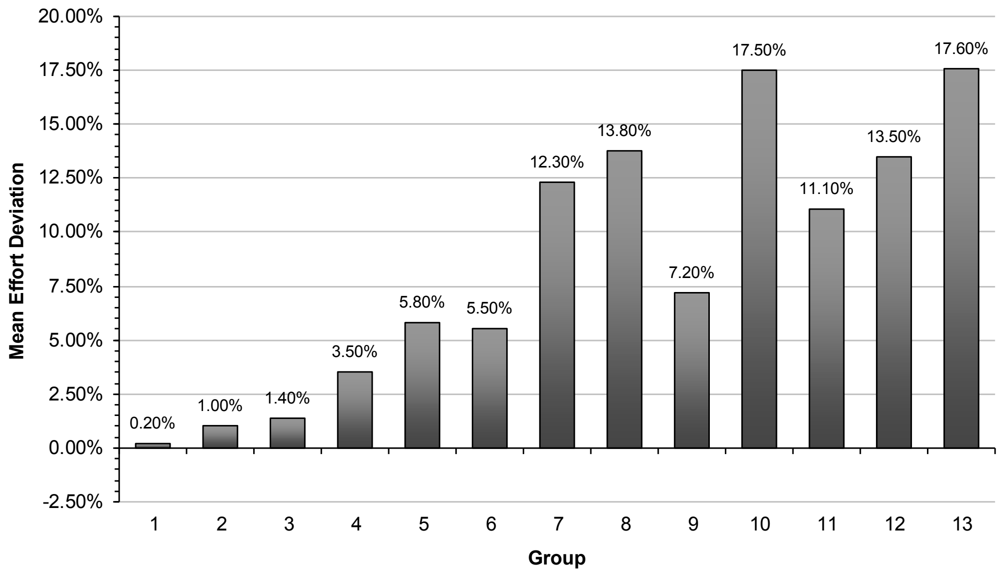


**Fig. 4.12:** Mean deviation of actual effort per group

In consequence, the given averages appear paradoxical and raise two questions:

1) Why are the estimates biased?

2) Does the presented estimation approach work in general?

Concerning the first question, the average efforts per exercise have evidently become invalid when drawing subsamples. The figures in Tab. 4.11 show that the population effort averages are *not* in line with the group averages (see column "mean effort deviation").[622] The groups 5-13 notably perform *below* average with mean effort overruns between 5.5% and 17.2% (see Tab. 4.11 and Fig. 4.12). This observation suggests that solution capability does not necessarily correlate with individual working speed. A correlation analysis of working speed and solution capability confirmed this suggestion (see Tab. 4.12; $r=-0.156$; $p<0.05$).

**Tab. 4.12:** Correlation of individual working speed and solution capability

| | | | Working Speed | Solution Capability |
|---|---|---|---|---|
| Spearman's rho | Working Speed | Correlation Coefficient | 1,000 | -,156* |
| | | Sig. (2-tailed) | . | ,048 |
| | | N | 162 | 162 |
| | Solution Capability | Correlation Coefficient | -,156* | 1,000 |
| | | Sig. (2-tailed) | ,048 | . |
| | | N | 162 | 162 |

*. Correlation is significant at the 0.05 level (2-tailed).

Students who ambitiously tried to solve as many exercises as possible without cheating or skipping exercises have a lower individual working speed than the entire population. The negative correlation and the given interpretation explain *why* the estimates are biased. Besides, with respect to real-world projects, this finding shows that it is important to assert that averages based on historical data are still valid when staffing new development teams as subsets of the entire developer pool.[623]

In order to answer the second question, that is, whether or not the presented estimation approach works in general, the approach must be statistically analyzed. KITCHENHAM and LINKMAN argue:

> *"When you are developing an estimation model (or process) you should always check your model's performance against past data. To assess whether your estimation model needs a correction factor, calculate the residual for each of your past projects. Residuals are the deviation between the value predicted by the estimation model and the actual value, that is, $r_i = y_i - y_i(est)$ where $y_i$ is the actual effort or duration, $y_i(est)$ is the estimated effort or duration, and $r_i$*

[622] A related statistical effect is known as *SIMPSON'S Paradox* or *YULE-SIMPSON Effect*, respectively (see SIMPSON (1951); WAGNER (1982)). This paradox refers to situations in which statistical analyses might yield inconsistent results. When comparing two populations, each split into groups, the observed incidences per group might become *reverse* after *combining* the groups. For example, a population is split into two groups, each showing a *positive* growth rate. Paradoxically, the combination of both groups might show a *negative* growth. In this exploratory study, however, the YULE-SIMPSON effect does not apply, since the dataset is based on *one* population.

[623] With reference to the individual differences paradigm, this issue was already addressed in Section 3.1.3.7. If individual differences are ignored, the computation and usage of averages for a group of different individuals might lead to inappropriate figures. E.g., STERNBERG, BERG (1992), p. 45.

*is the residual for the $i^{th}$ project in the dataset. If your estimation model is well-behaved, the sum of the residuals will be approximately zero. If the sum is not close to zero, your estimation model is biased and you must apply a correction factor."*[624]

The vague condition *"approximately zero"* can be specified. The sum of all residuals is significantly *not* different from zero if the mean of the residuals is less than the *standard error* (SE) of the mean residual.[625] Given the standard deviation $s$ of all $n$ residuals, the standard error of the mean is determined as follows:

$$SE = \frac{s}{\sqrt{n}} \qquad \textbf{Eq. 4.5}$$

The mean effort deviations per group already revealed that the estimates are biased (see Tab. 4.11 and Fig. 4.12). The corresponding analysis of residuals confirms this bias. Except for group 1, the mean of the residuals is greater than the standard error, and, therefore, the sum of residuals is different from zero (see Tab. 4.13).

**Tab. 4.13:** Standard error per group

| Group | Standard error | Mean of residuals |
|---|---|---|
| 1 | 4.97 | 2.60 |
| 2 | 6.97 | 10.46 |
| 3 | 5.42 | 14.20 |
| 4 | 8.29 | 50.86 |
| 5 | 7.84 | 85.71 |
| 6 | 6.60 | 73.14 |
| 7 | 7.33 | 200.22 |
| 8 | 6.69 | 231.29 |
| 9 | 8.16 | 139.26 |
| 10 | 8.67 | 366.93 |
| 11 | 9.71 | 251.11 |
| 12 | 10.04 | 311.39 |
| 13 | 11.07 | 455.11 |

In order to remove the bias, a group-specific correction factor is applied, which takes into account that higher groups have a slower working speed. Since we already know the group-specific mean deviation of effort (see Tab. 4.11), this information is used to adjust the estimates. For each group, the estimated effort is simply increased by the corresponding mean effort deviation percentage. In consequence, the mean of the residuals decrease. Except for group 13, the mean of the residuals is less than the standard error, and, for that reason, the sum of residuals is *not* different from zero.

[624] KITCHENHAM, LINKMAN (1997), p. 73.
[625] KITCHENHAM, LINKMAN (1997).

**Tab. 4.14:** Standard error per group (corrected estimates)

| Group | Standard error | Mean of residuals |
|---|---|---|
| 1 | 4.96 | 0.47 |
| 2 | 6.90 | 0.33 |
| 3 | 5.34 | 0.62 |
| 4 | 8.00 | 2.69 |
| 5 | 7.41 | 1.07 |
| 6 | 6.26 | 2.60 |
| 7 | 5.86 | 3.02 |
| 8 | 5.86 | 3.86 |
| 9 | 7.61 | 4.41 |
| 10 | 7.35 | 4.41 |
| 11 | 8.73 | 4.55 |
| 12 | 8.83 | 4.04 |
| 13 | 9.48 | 19.88 |

Finally, the corrected and uncorrected estimates can be compared. KITCHENHAM *et al.* summarize how they conducted such a comparison:

> *"To test whether the estimates obtained from one estimation model were significantly better than the estimates obtained from another, we used a paired t-test of the difference between the absolute residual for each model."*[626]

Since the residuals of some groups are not normally distributed, a non-parametric *WILCOXON Signed Rank Test* is conducted.[627] This test does not make the assumption of normality, and represents the non-parametric equivalent of the recommended paired t-test.[628] Similar to the paired t-test, the *WILCOXON* test compares the means of two groups. The *null hypothesis* is that the two means are equal. The test results show that the means are significantly different (p<0.001 for all 13 groups).[629] Thus, the null hypothesis must be rejected. Consequently, the presented approach does generally work, and can produce unbiased estimates.

In the following, however, the exploratory study does not use the group-specific correction factors, since they require *omniscience* about all potential development teams and their deviation of estimated and actual effort. This knowledge is considered unavailable for real-world development projects.

### 4.3.5 Analysis of Estimation Accuracy

The Bootstrap Dataset can be utilized to explore the possibilities and limitations of estimation accuracy. This subsection follows KITCHENHAM *et al.* and their helpful recommendations on estimation accuracy assessment.[630] In their paper *"What accu-*

[626] KITCHENHAM *et al.* (2002a), p. 63. See also KITCHENHAM, MENDES (2009); STENSRUD, MYRTVEIT (1998).
[627] WILCOXON (1945).
[628] E.g., PETT (1997).
[629] The same result was obtained by the paired t-test (p<0.001).
[630] KITCHENHAM *et al.* (2001).

*racy statistics really measure,"* KITCHENHAM *et al.* discuss two typical statistics, i.e., the *Mean Magnitude of Relative Error* and *Pred(25).* Both statistics are common instruments for assessing the accuracy of predictive models.[631] Given the actual effort $a_i$ and the estimated effort $e_i$ for each project $i$, the MMRE is calculated as follows:

$$MMRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|a_i - e_i|}{a_i} \qquad \textbf{Eq. 4.6}$$

*Pred(m)* refers to the percentage of estimates that are within *m%* of the actual values. Usually, accuracy is assessed by setting *m* to 25.[632] Pred(25), for example, is calculated by counting the number of estimates that match the actual efforts with a tolerance of 25%.

Following KITCHENHAM *et al.,* a new variable $z_0$ is introduced which results from dividing the estimated effort *e* by the actual effort *a*.[633]

$$z_0 = \frac{e}{a} \qquad \textbf{Eq. 4.7}$$

If $z_0$ is equal to 1.0, the exercises were solved on time. Values less than 1.0 indicate that the exercises were solved slower than expected, values greater than 1.0 refer to fast solutions. In order to get a first overview of how the estimations deviate, the following Fig. 4.13 visualizes the relationship of expected and actual effort of the 1st group based on $z_0$.

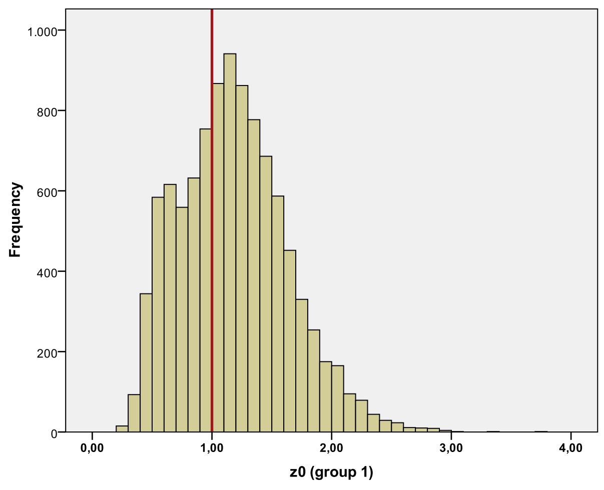


**Fig. 4.13:** Distribution of $z_0$ for the 1st group

[631] CONTE *et al.* (1986).
[632] KITCHENHAM *et al.* (2001).
[633] KITCHENHAM *et al.* (2001). The index *0* refers to the applied PARKINSON intensity. The corresponding PARKINSON effect is introduced later in this chapter.

Fig. 4.13 illustrates that the distribution of $z_0$ (group 1) is slightly *right-skewed*. Furthermore, the distribution shows that it is improbable for a single project to be exactly on time.

As another example, Fig. 4.14 shows the distribution of $z_0$ for the 10th group. This distribution has a similar spread, a higher peakedness while still being right-skewed. Besides, Fig. 4.14 visualizes that it is much more probable to have cost *overruns* of 50% (left of 1.0) than equivalent *underruns* (right of 1.0).

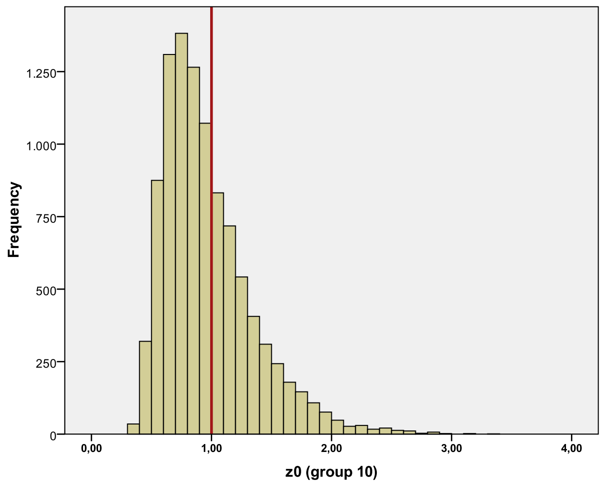


**Fig. 4.14:** Distribution of $z_0$ for the 10th group

In order to assess the estimation accuracy, it is beneficial to analyze both the *spread* and *kurtosis* of the estimator.[634] KITCHENHAM *et al.* demonstrate that *MMRE* is a measure of the spread and *Pred(m)* is a measure of the kurtosis of $z_0$. The values of *MMRE*, *Pred(25)*, as well as the standard deviation of all 13 groups can be found in the following table Tab. 4.15.

[634] KITCHENHAM *et al.* (2001). *Kurtosis* is a measure of the *"peakedness"* of a distribution.

**Tab. 4.15:** MMRE and Pred(25) of all groups (basic estimation kernel)

| Group | MMRE | Pred(25) | Pred(25) Underestimated | Pred(25) Overestimated | Standard Deviation |
|---|---|---|---|---|---|
| 1 | 0.38 | 39.28% | 19.01% | 41.71% | 0.47 |
| 2 | 0.44 | 35.99% | 17.08% | 46.93% | 0.69 |
| 3 | 0.32 | 48.54% | 14.67% | 36.79% | 0.52 |
| 4 | 0.47 | 31.48% | 22.53% | 45.99% | 0.59 |
| 5 | 0.41 | 36.70% | 23.83% | 39.47% | 0.53 |
| 6 | 0.34 | 44.36% | 21.52% | 34.12% | 0.47 |
| 7 | 0.33 | 42.83% | 28.91% | 28.26% | 0.45 |
| 8 | 0.29 | 50.63% | 28.07% | 21.30% | 0.40 |
| 9 | 0.33 | 47.10% | 23.85% | 29.05% | 0.41 |
| 10 | 0.30 | 48.44% | 32.61% | 18.95% | 0.41 |
| 11 | 0.32 | 47.05% | 26.44% | 26.51% | 0.43 |
| 12 | 0.32 | 46.12% | 28.99% | 24.89% | 0.43 |
| 13 | 0.29 | 46.75% | 33.35% | 19.90% | 0.41 |
| **Mean** | **0.35** | **43.48%** | **24.68%** | **31.84%** | **0.48** |

Interestingly, the given values of *MMRE* and *Pred(25)* that are based on the *Bootstrap Dataset* are in a range that is also plausible for real-world development projects. The values calculated by KITCHENHAM *et al.* using an empirical project dataset range from 37% to 42% for *Pred(25)* and from 0.599 to 0.697 for *MMRE*.[635] The accuracy statistics of all 13 groups refer to similar spreads (0.29 to 0.47) and peakednesses (31.5% to 50.6%), which gives evidence that the Bootstrap Dataset works as an appropriate basis for exploration.

The figures in Tab. 4.15 (*MMRE/Pred(25)*) highlight the importance of having knowledge of the distribution of actual and estimated effort, instead of working with just one single value representing the arithmetic mean. The knowledge of both the *spread* and the *kurtosis* provides a more detailed view and allows assessing the project risk.

For example, the distribution of $z_0$ of the 10th group given in Fig. 4.14 shows that it is possible to finish a project in half the time (2.0 on the x-axis). However, since the distribution is right-skewed, the probability of facing cost overruns of 100% (0.5 on the x-axis) is higher. The average cost overrun of 18% for the 10th group (see Tab. 4.11) can be interpreted as a warning for this.

Finally, concluding that a safety buffer of 20% eliminates project risk is dangerous. Tab. 4.15 shows that *Pred(25)* is only 48.44% for the 10th group, whereas 32.61% of the cases are underestimated. In view of that, the probability to overrun estimated effort by more than 20% is around 32%. Moreover, safety buffers bear the risk that

635 KITCHENHAM *et al.* (2001).

developers misapply the given buffers to reduce working speed. This behavior became known as the PARKINSON effect.[636]

### 4.3.6 Applying PARKINSON'S Law

***Theoretical Background***

In 1955, the British historian C. NORTHCOTE PARKINSON published a humorous article in *The Economist*, which was a satire of government bureaucracies largely based on his personal experience at the British Civil Service. In this article, PARKINSON describes how people organize their working time, which quickly became well-known as PARKINSON'S Law:

> *"Work expands so as to fill the time available for its completion."*[637]

PARKINSON illustrates the basic idea of this law by comparing an elderly lady, who invests an entire day writing a postcard to her niece, with a busy man, who needs three minutes for the same task. PARKINSON argues:

> *"Granted that work (and especially paperwork) is thus elastic in its demands on time, it is manifest that there need be little or no relationship between the work to be done and the size of the staff to which it may be assigned. [...] Politicians and taxpayers have assumed [...] that a rising total in the number of civil servants must reflect a growing volume of work to be done. Cynics, in questioning this belief, have imagined that the multiplication of officials must have left some of them idle or all of them able to work for shorter hours. [...] The fact is that the numbers of the officials and the quantity of the work to be done are not related to each other at all. The rise in the total of those employed is governed by Parkinson's Law, and would be much the same whether the volume of the work were to increase, diminish, or even disappear."*[638]

When agreeing on PARKINSON'S Law, the information about the allowed time has an effect on the expended time (see Fig. 4.15). If the law applies, a developer, for example, is unlikely to underrun the estimated time when completing a work package. In this context, the metaphor of *Gold-Plating* is often used:

636 PARKINSON (1955).
637 PARKINSON (1957), p. 13.
638 PARKINSON (1955), p. 2.

*"Developers are fascinated by new technology and are sometimes anxious to try out new features of their language or environment or to create their own implementation of a slick feature they saw in another product – whether or not it's required in their product. The effort required to design, implement, test, document, and support features that are not required lengthens the schedule."*[639]

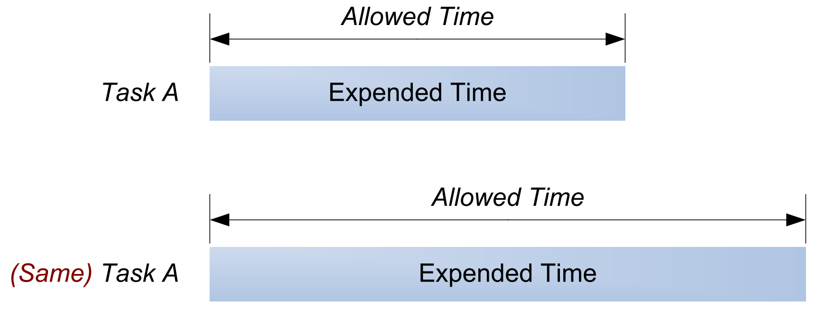


**Fig. 4.15:** The impact of PARKINSON'S Law on working speed

GOLDRATT'S *Student Syndrome* is another related phenomenon that focuses on how individuals manage available time.[640] The student syndrome describes that if individuals are given too much time, for example, due to safety buffers, they will *procrastinate* until they get close to the given deadline. In other words: The lack of motivation is suddenly replaced by a lack of time. At this point, they start working while often getting into time problems so that they cannot finish on time. GOLDRATT illustrates the phenomenon of procrastination by a fictitious dialog between project managers:

*"First, fight for safety time. When you get it, you have enough time, so why hurry. When do you sit down to do it? At the last minute. That's human nature."*[641]

### *New Estimation Kernel*

In order to explore the impact of the PARKINSON effect, a new estimation kernel was implemented. This kernel implies that the developers are informed about the average efforts of the work packages. Moreover, it implies that developers conform to the given estimates in different intensities. Therefore, the actual effort of each exercise was calculated in five variants. If the actual effort is below the expected effort, the unused time was filled by 0%, 25%, 50%, 75%, and 100%, each percentage representing different intensities of the PARKINSON effect.

[639] MCCONNELL (1996), p. 47.
[640] GOLDRATT (1997); MCCONNELL (2006).
[641] GOLDRATT (1997), p. 54.

For example, exercise 4 has a mean effort of 154 seconds (see Tab. 4.6). Assuming that one developer solves the exercise in 90 seconds, the unused time is 64 seconds. In this case, we get the following five time measures:

$$\begin{aligned}
Parkinson_o &= 90+64*0.00 = 90\\
Parkinson_{25} &= 90+64*0.25 = 106\\
Parkinson_{50} &= 90+64*0.50 = 122\\
Parkinson_{75} &= 90+64*0.75 = 138\\
Parkinson_{100} &= 90+64*1.00 = 154
\end{aligned} \qquad \textbf{Eq. 4.8}$$

For *Parkinson₀* the effect does not apply. This measure is always equal to the actual effort. Conversely, for *Parkinson₁₀₀* the effect consumes the entire deviation of expected and actual time. Therefore, this measure is always equal to the expected time.

### *Impact on Effort*

The overall influence of the different intensities of the PARKINSON effect on effort is summarized in Tab. 4.16.

**Tab. 4.16:** Mean effort deviation for different PARKINSON effect intensities

| Group | PARKINSON 0% | PARKINSON 25% | PARKINSON 50% | PARKINSON 75% | PARKINSON 100% |
|---|---|---|---|---|---|
| 1 | 100% | 107% | 113% | 119% | 126% |
| 2 | 101% | 109% | 117% | 125% | 133% |
| 3 | 101% | 108% | 115% | 121% | 128% |
| 4 | 104% | 112% | 121% | 129% | 138% |
| 5 | 106% | 114% | 122% | 130% | 138% |
| 6 | 105% | 113% | 120% | 128% | 135% |
| 7 | 113% | 120% | 127% | 134% | 141% |
| 8 | 114% | 121% | 127% | 134% | 141% |
| 9 | 107% | 115% | 123% | 131% | 139% |
| 10 | 118% | 125% | 132% | 139% | 146% |
| 11 | 111% | 119% | 126% | 134% | 142% |
| 12 | 114% | 121% | 129% | 136% | 144% |
| 13 | 117% | 124% | 132% | 139% | 147% |
| **Mean** | **109%** | **116%** | **123%** | **131%** | **138%** |

If the PARKINSON effect applies for real development projects, it must be taken into account when explaining effort variance. While the intensity of the effect might differ across developers and companies, the effect seems applicable to software development, since software is invisible (see Section 5.1.4.4), which generally makes tracking progress difficult, and work packages are often based on abstract and even incomplete information.

In this case, the expected time might be a helpful indicator for developers how to understand task descriptions in terms of scope. If this is true, the PARKINSON effect can be interpreted in two ways: 1) in its *unconstructive* form, developers adjust their working speed to fill unused time, and 2) in its *constructive* form, developers use effort estimates to conform their interpretation of work packages. In both cases, however, the PARKINSON effect can turn effort estimates into a *self-fulfilling prophecy*.

Interestingly, the mean effort overruns of $Parkinson_{75}$ are around 30%, which is in line with reported mean effort overruns of software development projects (see Section 3.1.3.1). According to a *WILCOXON Signed Rank Test*, the deviations based on the impact of the simulated PARKINSON effect are not random but statistically significant for all groups ($p<0.001$). As mentioned above, the accuracy statistics *MMRE* and *Pred(25)* of the Bootstrap Dataset are in a range that is also plausible for real-world development projects.[642] This finding, however, cannot be used to hypothesize that the PARKINSON effect alone explains the effort overruns of real-word projects, since there are other influences like optimism bias that can cause effort overruns. Nevertheless, the figures in Tab. 4.16 give evidence that the PARKINSON effect leads to plausible effort increases. Therefore, the effect is a candidate for explaining effort overruns in software development projects.

***Impact on Estimation Accuracy***

In order to analyze the PARKINSON effect's impact on estimation accuracy, four new variables are introduced that reflect the different intensities of the PARKINSON effect:

$$z_{25} = \frac{e}{a_{25}} \qquad z_{50} = \frac{e}{a_{50}} \qquad z_{75} = \frac{e}{a_{75}} \qquad z_{100} = \frac{e}{a_{100}} \qquad \textbf{Eq. 4.9}$$

The variables $a_{25}$, $a_{50}$, $a_{75}$, and $a_{100}$ represent the actual effort assuming different intensities of the PARKINSON effect. They correspond with the variables $Parkinson_{25}$, $Parkinson_{50}$, $Parkinson_{75}$, and $Parkinson_{100}$ (cp. Eq. 4.8). The resulting accuracy statistics are summarized in Tab. 4.17. It must be pointed out that an analysis of estimation accuracy is not superfluous due the previously observed increase of mean effort (see Tab. 4.16). While an effect increases mean effort, it might still have a positive impact on estimation accuracy.

In comparison to the zero intensity figures given in Tab. 4.15 (corresponds with $Parkinson_0$), the spreads of higher PARKINSON intensities (>0) described by *MMRE* gen-

[642] KITCHENHAM *et al.* (2001).

erally declined (see Tab. 4.17). The average of *Pred(25)* increased from 45% to 60%, which is mainly based on previously overestimated projects that now move into the 25% tolerance range. The percentages of *underestimated* projects slightly increased. Generally, for a PARKINSON intensity of 100%, the possibility of *overestimating* a project is not given (0%), because all developer fill the available time so that effort underruns are impossible.

**Tab. 4.17:** Estimation accuracy statistics of different intensities of the PARKINSON effect

| Group | PARKINSON 50% | | | | PARKINSON 100% | | | |
|---|---|---|---|---|---|---|---|---|
| | MMRE | Pred(25) | Pred(25) "under" | Pred(25) "over" | MMRE | Pred(25) | Pred(25) "under" | Pred(25) "over" |
| 1 | 0.215 | 64.91% | 21.50% | 13.59% | 0.150 | 74.53% | 25.47% | 0.00% |
| 2 | 0.234 | 58.99% | 22.01% | 19.00% | 0.170 | 71.56% | 28.44% | 0.00% |
| 3 | 0.188 | 72.76% | 19.24% | 8.00% | 0.168 | 73.62% | 26.38% | 0.00% |
| 4 | 0.255 | 52.98% | 28.31% | 18.71% | 0.204 | 64.70% | 35.30% | 0.00% |
| 5 | 0.241 | 56.25% | 29.99% | 13.76% | 0.219 | 61.41% | 38.59% | 0.00% |
| 6 | 0.209 | 65.24% | 27.38% | 7.38% | 0.213 | 64.46% | 35.54% | 0.00% |
| 7 | 0.235 | 56.60% | 37.43% | 5.97% | 0.247 | 51.87% | 48.13% | 0.00% |
| 8 | 0.223 | 59.15% | 37.37% | 3.48% | 0.255 | 49.87% | 50.13% | 0.00% |
| 9 | 0.218 | 61.32% | 32.77% | 5.91% | 0.247 | 53.06% | 46.94% | 0.00% |
| 10 | 0.244 | 53.18% | 43.37% | 3.45% | 0.285 | 40.66% | 59.34% | 0.00% |
| 11 | 0.227 | 59.82% | 35.07% | 5.11% | 0.256 | 51.37% | 48.63% | 0.00% |
| 12 | 0.234 | 57.55% | 37.57% | 4.88% | 0.269 | 48.24% | 51.76% | 0.00% |
| 13 | 0.242 | 53.88% | 43.62% | 2.50% | 0.292 | 41.00% | 59.00% | 0.00% |
| **Mean** | **0.230** | **59.43%** | **31.97%** | **8.60%** | **0.230** | **57.41%** | **42.59%** | **0.00%** |

To summarize, the mean efforts per group increase with higher PARKINSON intensities (see Tab. 4.16). This is, because a smaller number of "fast" work packages, which were completed ahead of schedule, are available to compensate "slow" work packages that exceeded the estimated effort. At the same time, the effect improves estimation accuracy to some extent. Due to the higher proportion of underestimated projects, it is questionable, however, whether or not to give this situation preference. Therefore, the PARKINSON effect (alone) just increases effort, but it does not clearly improve estimation accuracy.

At this point, it must be explained that the PARKINSON effect allows trading effort for estimation accuracy. If the project manager lets developers fill available time, the effort increases on average, since "fast" work packages converge to the estimates. This setting increases overall effort. In opposition, "slow" work packages that cannot be completed within the given timebox are unaffected by the PARKINSON effect. In consequence, the deviation of overall effort diminishes, so that accuracy statistics like *MMRE* and *Pred(25)* certify higher estimation accuracy. Yet, this higher accuracy is achieved at the expense of higher effort.

Generally, project managers can use safety buffers to lower the chance of effort overruns. Safety buffers can be based on a fixed percentage, e.g., 30%, or on surcharging multiples of the standard deviation of actual effort. Project managers are advised not to inform developers about these buffers, in order to prevent that they are consumed by the PARKINSON effect. If one project ends with effort overruns, such "secret" buffers can be used to compensate the deviation from the estimate to a certain extent.

### 4.3.7 Impact of Project Control

Another aspect that is worth exploring is *project control*. We therefore assume that a project leader tracks progress and informs developers whether or not the project gets behind schedule. If the project is behind schedule, the project leader will encourage developers to work focused and concentrated in order to work as fast as possible. Conversely, if the project is ahead of schedule, developers will work without pressure and the PARKINSON effect might apply.

#### *Estimation Kernel*

This trivial form of project control can easily be implemented as another estimation kernel. For each project (iteration), the actual work time $a_i$ and the estimated work time $e_i$ of the yet completed work packages (denoted by the index $i$) are accumulated. The cumulated values are used to determine the pressure of time $pr$ on which the project manager makes decisions:

$$pr = \frac{\sum_{i=1}^{15} a_i}{\sum_{i=1}^{15} e_i} \qquad \textbf{Eq. 4.10}$$

If the cumulated actual work time is less than the cumulated estimated time (pr<1.00), the project is ahead of schedule. In this case, the developers work as usual and the PARKINSON effect might apply. Conversely, if the cumulated work time exceeds the cumulated estimated time by 20% (pr>1.20), developers are put under pressure by project lead, so that they temporarily increase their working speed, decreasing actual effort by 20%. The pressure $pr$ is evaluated after each work package or exercise, respectively.

#### *Impact on Estimation Accuracy*

The impact on estimation accuracy is visualized in Fig. 4.16 for different PARKINSON intensities denoted by $z_0$, $z_{50}$, and $z_{100}$.

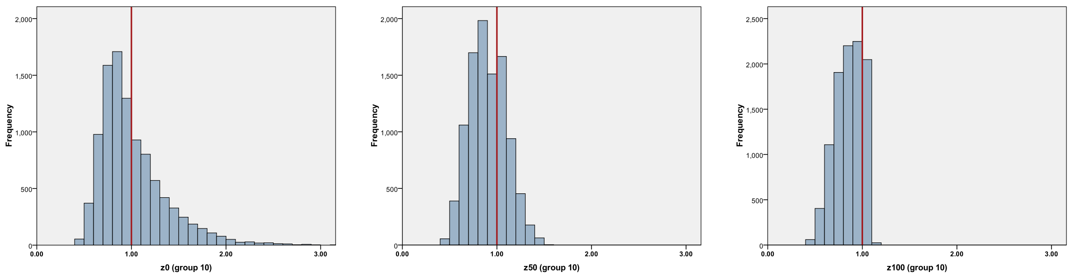


**Fig. 4.16:** Distribution of $z_0$, $z_{50}$, and $z_{100}$ of the 10th group under project control

For $z_0$, the effect of project control is marginal (see Fig. 4.16; left diagram; cp. Fig. 4.14), since project control can only speed up development by 20% if the team gets behind schedule. Conversely, the effect is stronger under the PARKINSON effect, because project control can allow or forbid filling available time. The histograms of $z_{50}$ and $z_{100}$ visualize that the distributions become more peaked while moving leftwards compared to $z_0$ (see Fig. 4.16; middle and right diagram). The corresponding statistics of spread and kurtosis are given in the following Tab. 4.18.

**Tab. 4.18:** Estimation accuracy statistics under simulated project control

| Group | PARKINSON 50% | | | | PARKINSON 100% | | |
|---|---|---|---|---|---|---|---|
| | MMRE | Pred(25) | Pred(25) “under” | Pred(25) “over“ | MMRE | Pred(25) | Pred(25) “under” |
| 1 | 0.181 | 72.51% | 13.74% | 13.75% | 0.087 | 86.03% | 13.97% |
| 2 | 0.200 | 67.83% | 12.68% | 19.49% | 0.095 | 86.81% | 13.19% |
| 3 | 0.155 | 81.69% | 10.06% | 8.25% | 0.088 | 89.57% | 10.43% |
| 4 | 0.222 | 60.95% | 19.19% | 19.86% | 0.124 | 79.74% | 20.26% |
| 5 | 0.201 | 66.01% | 19.47% | 14.52% | 0.126 | 79.49% | 20.51% |
| 6 | 0.171 | 75.78% | 15.97% | 8.25% | 0.116 | 83.18% | 16.82% |
| 7 | 0.183 | 71.54% | 21.67% | 6.79% | 0.139 | 77.07% | 22.93% |
| 8 | 0.165 | 76.78% | 19.20% | 4.02% | 0.136 | 79.62% | 20.38% |
| 9 | 0.171 | 74.70% | 18.04% | 7.26% | 0.129 | 80.55% | 19.45% |
| 10 | 0.178 | 72.48% | 23.11% | 4.41% | 0.155 | 75.42% | 24.58% |
| 11 | 0.175 | 73.95% | 20.17% | 5.88% | 0.139 | 78.27% | 21.73% |
| 12 | 0.178 | 71.87% | 22.32% | 5.81% | 0.146 | 76.20% | 23.80% |
| 13 | 0.176 | 72.27% | 24.26% | 3.47% | 0.155 | 74.42% | 25.58% |
| **Mean** | **0.181** | **72.18%** | **18.45%** | **9.37%** | **0.126** | **80.49%** | **19.51%** |

A comparison of Tab. 4.17 and Tab. 4.18 clearly shows that the spread (*MMRE*) decreased while the peakedness (*Pred(25)*) of the estimations increased. Especially under a full PARKINSON effect, 74% to 90% of the estimations matched the actual working time using a tolerance of 25% (see “Pred(25)” of “Parkinson 100%” in Tab. 4.18). For that reason, we can note that the PARKINSON effect has a positive impact on estimation accuracy when exercising forms of project control. At the same time, however, the effect still increases overall effort (see Tab. 4.19).

**Tab. 4.19:** Impact of the PARKINSON effect on overall effort under project control

| | **PARKINSON intensity** | | | | |
|---|---|---|---|---|---|
| **Group** | **0%** | **25%** | **50%** | **75%** | **100%** |
| 1 | 94.00% | 99.00% | 104.00% | 108.00% | 112.00% |
| 2 | 95.00% | 101.00% | 107.00% | 112.00% | 116.00% |
| 3 | 96.00% | 101.00% | 106.00% | 110.00% | 113.00% |
| 4 | 97.00% | 103.00% | 109.00% | 115.00% | 119.00% |
| 5 | 99.00% | 105.00% | 110.00% | 115.00% | 118.00% |
| 6 | 99.00% | 104.00% | 109.00% | 113.00% | 116.00% |
| 7 | 105.00% | 109.00% | 113.00% | 116.00% | 119.00% |
| 8 | 106.00% | 110.00% | 113.00% | 116.00% | 118.00% |
| 9 | 101.00% | 106.00% | 110.00% | 114.00% | 116.00% |
| 10 | 109.00% | 113.00% | 116.00% | 119.00% | 121.00% |
| 11 | 104.00% | 109.00% | 113.00% | 116.00% | 118.00% |
| 12 | 106.00% | 110.00% | 114.00% | 117.00% | 120.00% |
| 13 | 108.00% | 112.00% | 115.00% | 118.00% | 120.00% |
| **Mean** | **101.46%** | **106.31%** | **110.69%** | **114.54%** | **117.38%** |

In view of the presented ratios, empirical observations of good estimation accuracy must be critically challenged. On the one hand, good estimation results might be truly based on good estimation techniques. On the other hand, high estimation accuracy might also be based on overestimation while developers fill the available time according to PARKINSON'S Law.

Furthermore, it must be asked at this point, whether or not it is reasonable to assume that project leaders, who are responsible for completing projects on schedule, set up and launch a development project like an automated mechanical process in which they cannot intervene until completion. Are project leaders not tempted to do anything in their power so that the projects meet the schedule? Tab. 4.20 outlines how project leaders might mistakenly perceive their role in and impact on the software development process. If this perception applies for real-world development projects, we must expect this type of project leaders to intervene whenever possible. Generally, the independency of estimates and actual effort becomes questionable when taking the interests of project lead into account.

**Tab. 4.20:** Project leaders' potential perception of development projects

| Final project status | What happened? | Project leader's perspective |
|---|---|---|
| The project exceeded the schedule. | The project was underestimated, that is, the estimates were too low. | *"The developers did a bad job. They ignored my advices, ignored the process guidelines, and got lost in unnecessary details."* |
| The project ended on time. | 1) The project was underestimated but the developers did a brilliant job.<br>2) The project was overestimated and the developers filled the available time.<br>3) The estimates were accurate and the developers did a good, reliable job. | *"Well done! I did a perfect job."* |
| The project ended ahead of schedule. | The project was overestimated, that is, the estimates were too high. Yet, the developers did a good job and refrained from filling all the available time. | *"Finally, developers start to follow my advices and they conformed to the process guidelines. And guess what: we can complete projects ahead of schedule. I hope, they have learned from this lesson."* |

### 4.3.8 Consideration of Individual Differences

In order to minimize the PARKINSON effect, an overall reduction of the estimate, for example, by 30%, most probably leads to cost overruns, since *all* work packages will be affected by this reduction. That is, the overall reduction also cuts estimates that cannot be hold even without any PARKINSON effect. However, some work packages in a project allow reducing the estimated time (those which would be affected by PARKINSON), while other work packages demand more work time (those which would exceed the estimates). The challenge is selecting the *right* packages for either effort reduction or increase. In case of the simple random subsamples, the consideration of *individual differences* among the developers might help making this decision.[643]

In order to analyze the consideration of individual differences and its impact on overall effort and estimation accuracy, the previous estimation kernel was extended. The new estimation kernel uses the individual working speed of students to adjust the estimates. For "slow" developers (below average), the estimates are systematically increased, while "fast" developers (above average) are confronted with reduced esti-

[643] See Section 3.1.3.7.

mates. The corresponding figures for different PARKINSON intensities are given in Tab. 4.21.

**Tab. 4.21:** Impact of individual differences on effort and estimation accuracy

| | PARKINSON 0% | | PARKINSON 50% | | PARKINSON 100% | |
|---|---|---|---|---|---|---|
| **Group** | **Mean effort deviation** | **Pred(25)** | **Mean effort deviation** | **Pred(25)** | **Mean effort deviation** | **Pred(25)** |
| 1 | 98% | 74% | 105% | 92% | 111% | 98% |
| 2 | 98% | 73% | 106% | 92% | 112% | 96% |
| 3 | 98% | 83% | 105% | 96% | 109% | 97% |
| 4 | 99% | 70% | 107% | 89% | 112% | 91% |
| 5 | 101% | 76% | 109% | 90% | 114% | 92% |
| 6 | 102% | 77% | 111% | 93% | 117% | 96% |
| 7 | 108% | 80% | 114% | 92% | 118% | 92% |
| 8 | 110% | 81% | 117% | 95% | 121% | 95% |
| 9 | 104% | 72% | 113% | 94% | 118% | 97% |
| 10 | 112% | 77% | 119% | 90% | 123% | 91% |
| 11 | 106% | 82% | 111% | 91% | 114% | 91% |
| 12 | 107% | 83% | 112% | 90% | 115% | 90% |
| 13 | 112% | 75% | 121% | 93% | 126% | 94% |
| **Mean** | **104%** | **77%** | **112%** | **92%** | **116%** | **94%** |

In view of the figures in Tab. 4.20 compared to Tab. 4.18, the consideration of individual differences among developers allows improving estimation accuracy (cp. *Pred(25)*) while keeping overall effort constant (cp. "Mean effort deviation"). For the highest PARKINSON intensity, *Pred(25)* could be increased from 80% to 94% on average.

## 4.4 Summary

Although the Exercise Dataset is solely based on the learning activities of bachelor students, it allows drawing conclusions that might also be true for developers in real world software projects. To recapitulate, the data exploration led to the following findings:

- Students show different levels of solution capability (see Section 4.3.1). While simple exercises were solved by more than 80% of the students, the percentages quickly decline for more difficult exercises. Less than 25% were able to solve the last five exercises and less than 10% solved the last two exercises. Such differences are in line with the individual differences paradigm.

- Similar to solution capability, the solution effort also strongly varies across students (see Section 4.3.2). The effort ratios range from 1:12 to 1:97.
- Generally, the mean solution effort strongly correlates with the supposed level of difficulty ($\rho=0.850$; $p<0.01$; see Section 4.3.2).
- Students have an individual working speed, which is independent of the individual solution capability and independent of the programming language (see Section 4.3.3).
- Averages calculated on historical data can bias effort estimation when staffing new developer teams. Historical data is not necessarily valid for a particular team, and, thus, might lead to systematic errors in future estimation processes (see Section 4.3.4.2). Yet, in this study the bias could be removed. Accordingly, the presented approach does generally work, and allows producing unbiased estimates.
- Although the dataset allows perfect analogy for comparing work packages, effort estimates considerably deviate from the actual efforts when using averages.
- The accuracy statistics *MMRE* and *Pred(25)* of the *Bootstrap Dataset* are not contradictory to real-world development projects, sine they are in line with the statistics of empirical project datasets.
- If developers are influenced by given estimates and fill the available time, the effort increases by 30% on average (Section 4.3.6). The PARKINSON effect describes this behavior.
- Project control can reduce overall effort by limiting the PARKINSON effect. Consequently, project control can have a positive effect on estimation accuracy (Section 4.3.7).
- Considering individual differences of students (e.g., the average working speed) improves estimation accuracy without raising overall effort (Section 4.3.8).

Above and beyond, the data exploration led to a fundamental consideration. Given a particular distribution of effort in the dataset, it appeared impossible to excess a particular level of estimation accuracy. However, the achieved estimation accuracies were still unsatisfying, although it allowed perfect analogy of work packages. The distribution of *z*, for example, originates from the fact that the effort averages ignore or obscure the actual variability that is given in the data. The actual efforts still have a strong variability, which cannot be reflected by the estimates.

This poses the question, how one can overcome such difficulties in real world software projects, since effort estimation is additionally affected by uncertainty, environmental influences, incomplete historical data, vague forms of analogy etc. A potential answer is that effort estimates – once produced – influence the project staff, because it might be the only known and easily traceable parameter in the development project. This conclusion is supported by the findings of the exploratory study. Combinations of the PARKINSON effect and simple forms of project control have provided the best estimation accuracy. In this case, however, there is a feedback loop between effort estimation and project members, including both the management and the developer teams.

In consequence, it does not seem reasonable to assume a general independency of estimated and actual software development effort. Instead, effort estimates must be considered as self-fulfilling prophecies in real-world development projects. Accordingly, this exploratory study provides three fundamental hypotheses:

1) In order to achieve high estimation accuracy, project managers and developers must conform to given estimates.

2) Given high PARKINSON intensities, the distribution of effort hardly allows effort underruns, since work packages are either completed on time or behind schedule.

3) The consideration of individual differences among developers improves estimation accuracy.

A corresponding theory including a causal model, a set of assumptions and propositions, as well as general limitations is developed in the next chapter.

# 5 ESTIMATION FULFILLMENT THEORY

All size and effort estimation techniques presented in Chapter 2 have in common that they do not account for the effect of introducing estimates to project staff.[644] Those techniques address effort estimation as an isolated, independent problem assuming that the estimates do not influence themselves. However, the literature review, the discussion of the nature of software development, as well as the exploratory study indicated that the opposite *can* be true.[645]

Consequently, this chapter focuses on this controversy in order to construct a theory based on the observed phenomena. It has to be pointed out that the theory constructed and discussed in the following *can* apply for particular software development projects or even particular software organizations. The theory does not claim to be generally true for all types of development projects or organizations. Accordingly, a set of *assumptions* and *propositions* will be stated that can be tested empirically. Prior to this *predictive* component, the causal model of the theory is developed which represents the *explanatory* component of the theory.

The distinction of an explanatory and a predictive component is made with respect to GREGOR'S recommendations on IS theory.[646] Towards the question *what theory is*, GREGOR explains that:

> *"In general, philosophers of science writing in the tradition of the physical or natural sciences are likely to see theory as providing explanations and predictions and as being testable."*[647]

Correspondingly, POPPER'S view on theory is as follows:

> *"Scientific theories are universal statements. Like all linguistic representations they are systems of signs or symbols. Theories are*

[644] One exception is the *Integrated Dynamic Systems* approach. See Section 5.4.
[645] The literature review can be found in Chapter 2. The crux of software development is discussed in Chapter 3. The explorative study can be found in Chapter 4.
[646] GREGOR (2006).
[647] GREGOR (2006), p. 614.

> *nets cast to catch what we call 'the world'; to rationalize, to explain and to master it. We endeavor to make the mesh even finer and finer."*[648]

Although this characteristic view on theory is rooted in the natural sciences, it can also be applied to the social sciences.[649] However, some traditions originated from social sciences that can be opposed to natural sciences. The interpretivist tradition, for example, does not predominantly focus on testable theory but on understanding complex social phenomena from the individual's point of view.[650] In order to consolidate different views on theory, GREGOR recapitulates that:

> *"Different perspectives on theory at a general level shows theories as abstract entities that aim to describe, explain, and enhance understanding of the world and, in some cases, to provide predictions of what will happen in the future and to give a basis for intervention and action."*[651]

HATCH and CUNLIFFE define theory by differentiating academic theory from common sense, ideas, and expectations that can also be transferred into personal concepts.[652] They argue that:

> *"The basic difference between common sense theorizing and the theorizing academics do is the added care academics take to specify their practice, correct its errors and share their theories with others, thereby contributing to systematic knowledge-building efforts. Theories are built from abstractions known as concepts. One concept – called the phenomenon of interest – is selected from all the others as a focus for theorizing and then related concepts are defined and used to explain that one. Consider ALBERT EINSTEIN's theory that $E = mc^2$. Energy (E) was EINSTEIN's phenomenon of interest and he explained it using the concepts of mass (m) and a constant representing the speed of light (c). The squaring of c, and its multiplication by m, specify how these explanatory concepts are related to the phenomenon of interest and form EINSTEIN's theory about the relationship between energy and matter. In a nutshell, $E = mc^2$ shows what theory is – a set of concepts and the relation-*

648 POPPER (1980), p. 59.
649 GREGOR (2006).
650 GREGOR (2006).
651 GREGOR (2006), p. 616.
652 HATCH, CUNLIFFE (2006).

> *ships between them proposed to explain the phenomenon of interest."*[653]

With respect to IS research and the aspiration of IS researches, GREGOR argues that *"we need a language of our own to talk about theory and should not adopt uncritically ideas about what constitutes theory from any one other disciplinary area."*[654] Additionally, she advices that *"the nature of theory in itself is at least as important as domain, epistemological and sociopolitical questions, which to date have attracted a disproportionate share of the discussion of IS research."*[655]

The theory developed in this work follows GREGOR'S classification of and her recommendations on different theory types in IS research. The estimation fulfillment theory comprises an explanatory as well as a predictive component, and, therefore, is an EP-type theory.[656] Consequently, the theory describes constructs (*"Building Blocks"*), explain their causal relationships, and provides testable propositions under defined assumptions. The theory will respond to all relevant *"what is, how, why, when, and what will be"* questions. The aspects of theory *corroboration*, *falsification*, and *counter-findings* are introduced and discussed in the next chapter, which focuses on testing the theory by confronting it with empirical content.[657]

## 5.1 Explanatory Component

### 5.1.1 Building Block 1: Role of Project Lead

The primary objective of effort estimation is *budgeting*, followed by project planning and project control.[658] An estimate, however, is usually produced in a pre-planning phase long before the software is developed and further details are known. Despite the high degree of uncertainty at this point, the estimate becomes a means that is used for early negotiations with customers and management. The customer wants to know the price for building the software. Similarly, the management decides how much budget is approved for the project. In consequence, the most important stakeholders expect the project to be in line with the approved budget and start to put pressure on project lead (see Fig. 5.1).

653 HATCH, CUNLIFFE (2006), p. 5.
654 GREGOR (2006), p. 635.
655 GREGOR (2006), p. 635.
656 GREGOR (2006).
657 E.g., KUHN (1970); LEE (1991); POPPER (1959); POPPER (1980).
658 The objectives of effort estimation are briefly summarized in Section 2.3.

From this moment on, renegotiations by project lead with upper management or the customer must be regarded as impossible or at least difficult, because the project is subordinated to the financial interests of the stakeholders. Therefore, increasing the estimated effort or stretching the project schedule is not welcome. However, renegotiations with developers are less complicated since developers are subordinated to project lead. Accordingly, the effort of work packages can be adjusted or unclear requirements can be interpreted and planned in a way that suits the project schedule. In consequence, project lead both has an interest and means to make sure that the actual effort matches the approved effort estimate.

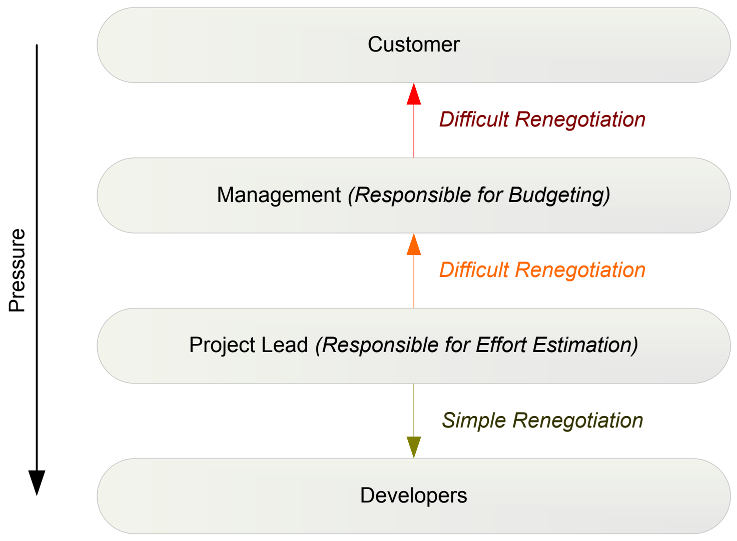


**Fig. 5.1:** Most likely direction of renegotiations

According to the literature review, the most common estimation techniques are estimation by analogy and expert judgment.[659] Moreover, studies on estimation accuracy found that estimation by analogy provides the most precise results. On the one hand, it can be assumed that estimation by analogy results in high estimation accuracy because it automatically accounts for company-specific factors, like average staff skill, influences of the development environment etc. On the other hand, it can be assumed that estimates produced by analogy and expert judgment become personally connected to the persons who made them. Besides, such estimation techniques do not require complete and detailed requirements. The estimates produced in the pre-planning phase might therefore leave room for interpretations on how to implement particular work packages. If more detailed information is available, for example, when writing the specifications of work packages, the produced estimates might have already become target values. If this is the case, project lead might not adjust the estimates but the specifications in order to suit the schedule.

[659] See Section 2.4.

When applying these considerations on the *project management triangle*, the estimated effort and especially the approved budget become almost immobile figures while project scope and software quality turn into moving targets (see Fig. 5.2). This is in contrast to the common supposed setting in which schedule and scope are fixed, and, for this reason, the only way to ensure a defined quality level is increasing effort and budget.

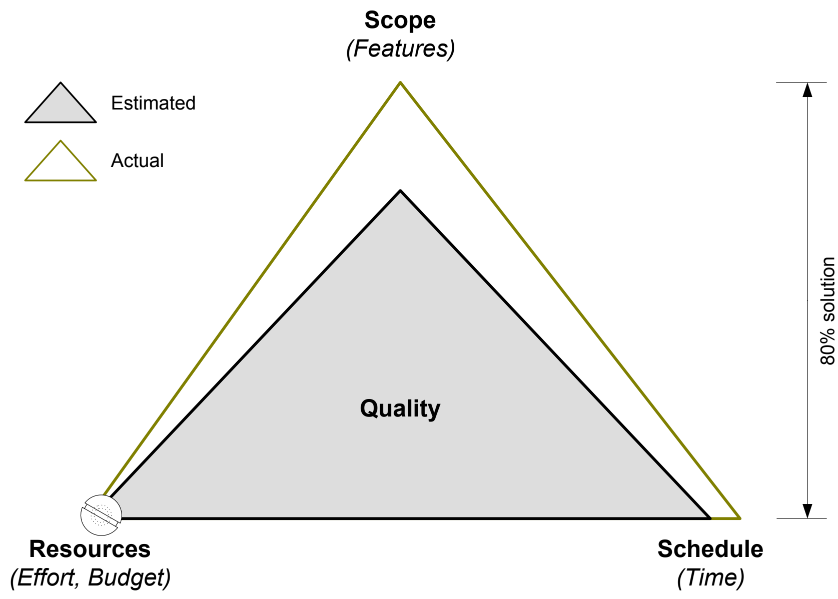


**Fig. 5.2:** Project management triangle

### 5.1.2 Building Block 2: Behavior and Competence of Developers

The PARKINSON effect and the *Student Syndrome* both explain what happens when individuals are confronted with deadlines.[660] Work time can generally be expanded to fill all available time. Similarly, individuals *procrastinate* by concentrating on other, unnecessary tasks until time pressure replaces the lack of motivation necessary for focusing on the critical task. Besides the PARKINSON effect and the Student Syndrome, two further effects are regarded as important in this context.

#### 5.1.2.1 Learning Curve Effect

The preliminary development of the *learning curve* is accredited to the German psychologist EBBINGHAUS.[661] The learning curve concept – as it is known today – was

[660] The PARKINSON effect and the *Student Syndrome* are described in Section 4.3.6.
[661] EBBINGHAUS (1885); LINDA *et al.* (1995).

initially published by WRIGHT in 1936.[662] During empirical studies on airplane production, WRIGHT observed a relationship between the cumulative quantity of manufactured units and the number of required labor. As the cumulative quantity doubles, the required labor decreases at a uniform rate, which became known as the *Learning Curve Effect*. The general learning curve can be expressed by the following equation:

$$Y = KX^{\log_2 \phi} \qquad \textbf{Eq. 5.1}$$

In this equation, $Y$ is the number of working hours required for producing the next unit, whereas $X$ is the cumulative number of previously produced units. $K$ represents the number of working hours required for the production of the first unit. Finally, $\Phi$ describes the learning rate, for example, 90%. A learning rate of 90% means that the working time decreases to 90% after the next doubling of the cumulative working hours. Due to the learning curve effect, productivity monotonically increases over time (Fig. 5.3; log-linear model).

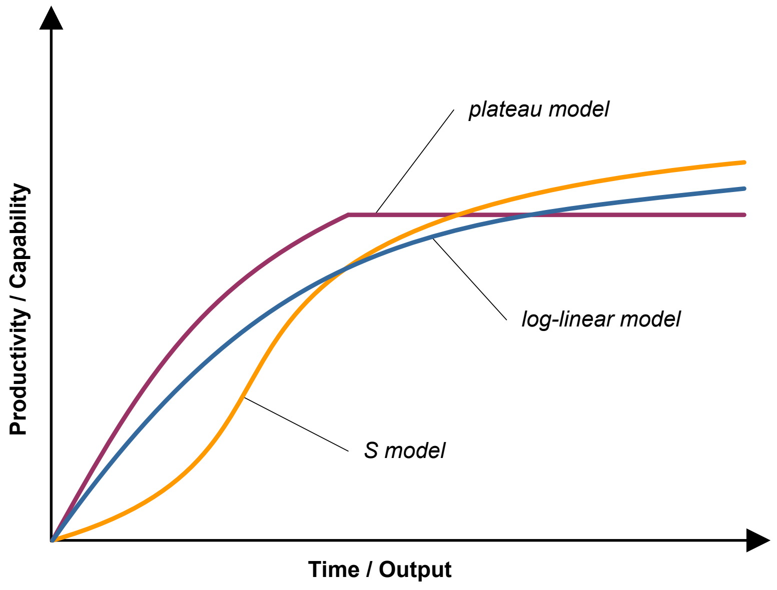


**Fig. 5.3:** Growth of productivity according to learning curve effects

According to YELLE, numerous geometric variants of the original, log-linear learning curve have been proposed in the past, especially the *Plateau Model* and the *S-Model* (see Fig. 5.4).[663] An overview of common learning curve variants is given by CARLSON.[664]

[662] WRIGHT (1936); YELLE (1979). Common synonyms of the *learning curve* are *progress curve*, *improvement curve*, and *experience curve*.

[663] YELLE (1979).

[664] CARLSON (1973); CARLSON (1976).

The phenomenon of *Plateauing* was observed subsequent to WRIGHT'S studies in different contexts of manufacturing. During a *startup* phase, the observed workers showed typical learning rates and improved their personal productivity. Later, however, workers entered a *steady-state* phase where the individual productivity remained constant.[665]

Based on empirical observations, COCHRAN developed an *S-shaped* learning curve. In this model, the learning curve starts with a *concave* form, later turning to a *convex* form, which reflects different learning rates over time.[666] COCHRAN assumes that workers have to familiarize with new tasks first. This familiarization causes low learning rates in the beginning. This phase, however, is followed by high learning rates since workers become more and more trained and develop routine. Later, the learning rates decrease similar to WRIGHT'S log-linear model.[667]

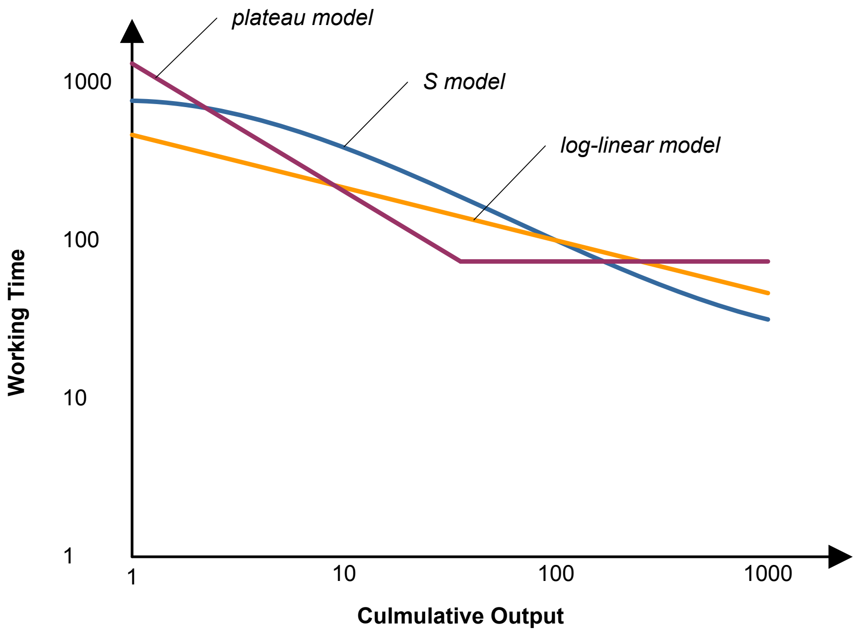


**Fig. 5.4:** Common learning curve models[668]

### 5.1.2.2 Conscious Competence Learning Model

The *Conscious Competence (Learning) Model* describes four stages each individual passes through when progressing from incompetence to competence with regard to a particular skill (see Fig. 5.5). The originator of the model is not known. CHAPMAN gives a detailed discussion of potential origins of the conscious competence mod-

---

[665] BALOFF (1966); BALOFF (1971); CONWAY, SCHULTZ (1959); YELLE (1979).

[666] COCHRAN (1960); KUNOW (2006).

[667] KUNOW (2006).

[668] YELLE (1979), p. 304.

el.[669] He also provides a comprehensive introduction. The conscious competence model is considered as a useful framework in order to understand and give a context for the DUNNING-KRUGER effect, which is described in the next subsection. The four stages of competence are briefly outlined in the following:[670]

1) *Unconscious incompetence*. The individual neither understands nor knows how to perform a task that requires the new skill. Additionally, the individual is not able or does not want to assess his or her deficit compared to skilled individuals.

2) *Conscious incompetence*. The only difference to the first stage is that the individual starts to recognize a deficit. He or she understands that the task to be performed requires a new skill.

3) *Conscious competence*. The individual made progress. He or she is able to perform the task based on the new skill. However, using the new skill requires consciousness and concentration.

4) *Unconscious competence*. The individual developed routine. He or she utilizes the skill unconsciously with little effort and concentration. The ability to teach the skill to others, however, depends on other personal factors.[671]

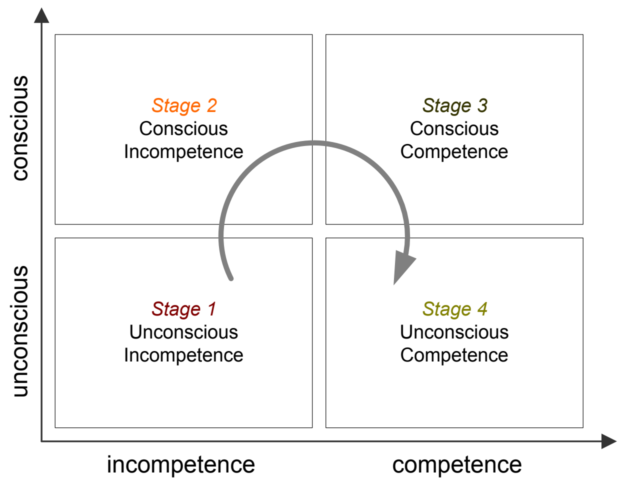


**Fig. 5.5:** Four stages of competence

[669] CHAPMAN (2009).
[670] E.g.. CLARKSON (2001); DICKMANN, STANFORD-BLAIR (2000).
[671] CHAPMAN (2009).

In order to demonstrate the basic concept of the *Conscious Competence Model*, CLARKSON gives a simple, illustrative example most of us might have experienced. CLARKSON speaks of steps instead of stages:

> *"When we start to use a computer keyboard we are in step 1, unaware that there are some keyboard skills we don't yet have. As we start to use the keyboard and try to type quickly we move into step 2, realizing that there's more to this than meets the eye. If we learn to type properly, we move into step 3 where we have to work hard at hitting the right keys without always looking at what we are doing, and getting it right more and more often. When we have mastered typing skills, then we are in step 4 where we don't have to think about where the keys are but just use them automatically."*[672]

### 5.1.2.3 DUNNING-KRUGER Effect

In 1999, DUNNING and KRUGER published their results of four psychological studies, themed *"Unskilled and Unaware of It."*[673] Their studies focused on how individuals self-assess their competence or incompetence respectively. DUNNING and KRUGER give three propositions of which they characterize the first two as noncontroversial by referring to numerous supportive, former studies: 1) In many domains, success and satisfaction depend on knowledge, wisdom, and knowing which rules to follow and which decision to make, 2) individuals, however, differ extensively in the knowledge and strategies they apply in these domains.[674] The third, more controversial proposition of DUNNING and KRUGER reads as follows:

> *"When people are incompetent in the strategies they adopt to achieve success and satisfaction, they suffer a dual burden: Not only do they reach erroneous conclusions and make unfortunate choices, but their incompetence robs them of the ability to realize it."*[675]

DUNNING and KRUGER argue that the skills that account for one's competence in a domain are also necessary to evaluate one's own or anyone else's competence in that particular domain. As a simple example, they refer to English grammar. The skills necessary to write correct sentences are the same skills necessary to evaluate whether a sentence is correct or not. They argue that:

[672] CLARKSON (2001), p. 5.
[673] DUNNING *et al.* (2003); DUNNING, KRUGER (1999).
[674] DUNNING, KRUGER (1999).
[675] DUNNING, KRUGER (1999), p. 1121.

> *“[...] incompetent individuals lack what cognitive psychologists variously term metacognition, metamemory, metacomprehension, or self-monitoring skills. These terms refer to the ability to know how well one is performing, when one is likely to be accurate in judgment, and when one is likely to be in error.”*[676]

The four conducted studies tested abilities in humor, logical reasoning, and grammar. Each test also asked for self-assessment. DUNNING and KRUGER proposed four predictions that were confirmed by the study results and the corresponding statistical analysis:[677]

1) Incompetent individuals will dramatically overestimate their ability and performance.

2) Incompetent individuals have a deficit in recognizing someone else’s competence.

3) Incompetent individuals will be unable to use information about the choices and performances of others. Therefore, they do not improve their self-assessment.

4) When developing competence, the incompetent individuals start to gain insight about their deficits.

In a nutshell, the *DUNNING-KRUGER effect* describes why incompetent individuals are likely to draw erroneous conclusions and pursue wrong strategies, while being unable to realize it.

This effect is regarded as a relevant influence on project performance and project risk as well.[678] If the effect applies for particular developers, they will be unable to realize their incompetence when they become overwhelmed by particular work packages. In consequence, they can neither evaluate their own performance nor the quality of their work appropriately. As a worst case, incompetent developers are assumed to spend the entire estimated work time until they get behind schedule. At this point, they cannot evaluate whether their solution is nearly complete or far away from completion. Moreover, they cannot assess the correctness of their solution. HARRISON gives a comment on the impact of the DUNNING-KRUGER effect on software development:

676 DUNNING, KRUGER (1999), p. 1121.
677 DUNNING, KRUGER (1999).
678 E.g., DUNNING (2006); HARRISON (2004); JORGENSEN (2004).

> *"We've probably all run into hotshot programmers who speak in bits and bytes, promise the moon, but somehow never seem to deliver their work products (at least ones that actually work) on time. I never really thought of this as some kind of universal pattern. Rather, I simply assumed that it was the luck of the draw, and I continued to take both organizational as well as personal self-assessments of capabilities at face value. After all, we're taught that self-confidence is important, and to show doubt in your capabilities is the equivalent of hanging a sign that says 'INCOMPETENT' around your neck. [...] If it were true that the important thing is gaining knowledge, the implications would be tremendous. Rather than actually changing their processes, organizations could simply study their processes to gain more knowledge. [...] Obviously, programmers gain a better understanding of what they do and how they do it if they actually track how they're spending their time."*[679]

#### 5.1.2.4 Impact on the Causal Model

If the PARKINSON effect applies for single developers or the whole development team, the working speed and work focus will be adjusted in a way that the actual time for completing a work package is close to the given estimate. This can happen either unconsciously or consciously.

First, novice developers might be affected by the DUNNING-KRUGER effect, and, therefore, they cannot assess their own work in terms of quality and completeness.[680] Moreover, novice developers are almost certainly on a low *learning plateau* and need more time to complete work packages compared to more experienced developers.[681] In consequence, they will invest the complete estimated time, being unconsciously affected by the PARKINSON effect.

Second, developers might consciously fill all available time because this behavior leads to relaxed workdays. In this case, developers might try to reach a high degree of completion as soon as possible and then slow down or turn to *gold-plating*.[682] Since developers are usually not awarded for being ahead of schedule, there might be

[679] HARRISON (2004), pp. 5-7.
[680] For the DUNNING-KRUGER effect see Section 5.1.2.3.
[681] The *Learning Curve Effect* is described in Section 5.1.2.1.
[682] *Gold-plating* is described within the context of the PARKINSON effect in Section 4.3.6.

no incentive not to fill the available time. This behavior refers to the *unconstructive* form of the PARKINSON effect because developers simply waste available time.[683]

Third, developers use the effort estimate as an essential component of the specification. Specifications are often incomplete, vague, and leave room for interpretations. On the other hand, good performance, high software quality, reliability, pretty graphical interfaces, good user experience etc. are always welcome. Therefore, developers will always find reasons to expand work time in good faith.

For example, one developer is assigned to a work package, which addresses the implementation of a web-based, statistical report of sales data. Assuming the effort estimation is *two* workdays. The developer will interpret the work specification in a way that suits a timebox of *two* workdays. He or she then implements the necessary database queries and codes a straightforward, table-based output generation.

Alternatively, we assume the effort estimate to be *four* workdays. In this case, the work specification can be interpreted differently. Accordingly, the developer creates a first draft of the solution. Then he or she might optimize the database query for better performance. Next, several different output styles are created, tested, and compared. Table columns of the report are made sortable. Finally, a feature is added that allows users to customize, which columns appear in the report.

This example illustrates the impact of knowing the available time. Since the effort estimate is produced by project lead while analyzing *what* to build, the estimate is again used by developers for answering the same question. This form of the PARKINSON effect has already been classified as the *constructive* form, since developers use effort estimates to conform their interpretation of work packages.[684]

Finally, procrastination and PARKINSON entail the risk of systematic cost overruns. Work packages that could have been completed ahead of schedule are completed on time, while work packages that are completed behind schedule increase overall effort. Even if the effort estimate is precise and feasible, the project will end with cost overruns.

Fig. 5.6 gives examples based on this thought. The first work package (WP 1) could be completed ahead of schedule. The responsible developer, however, fills the available time, so that he or she finishes on time (PARKINSON). For the second work pack-

[683] The PARKINSON effect can be interpreted in two ways: 1) in its *unconstructive* form, developers adjust their working speed to fill unused time, and 2) in its *constructive* form, developers use effort estimates to conform their interpretation of work packages. See Section 4.3.6.

[684] See Section 4.3.6.

age (WP 2), the developer procrastinates at the beginning. Since the developers needs the total estimated time for completion, the work package finally ends with effort overruns. The other examples demonstrate that combinations of PARKINSON and procrastination are possible (WP 3 and WP 4), that PARKINSON and procrastination might also be found in the middle of work packages (WP 6), and that work packages can exceed the estimates even without such effects (WP 5).

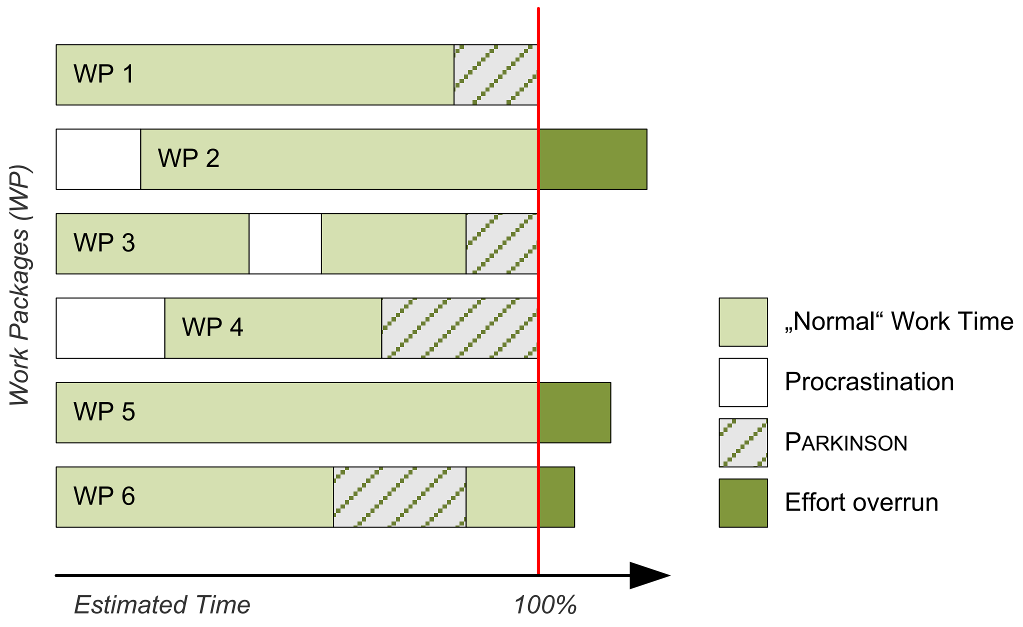


**Fig. 5.6:** Systematic cost overruns caused by PARKINSON and procrastination

## 5.1.3 Building Block 3: Craftsmanship of Skilled Individuals

Development process models, e.g., waterfall or iterative models, used technologies, programming languages, as well as development environments have an impact on overall effort.[685] To be more precise, such factors influence the productivity of the development team. Agile development methodologies, for example, favor communication and regular meetings in order to prevent information asymmetries.[686] A reduction of information asymmetries might lead to a smaller amount of misinterpreted requirements, which has a positive impact on the proportion of rework. In return, overall productivity of the team increases.

Effort estimation techniques usually take the environmental characteristics into account.[687] Especially when using expert estimation or estimation by analogy, specific

[685] See Section 3.1.3.5.
[686] See Section 3.1.2.3.
[687] See, for example, the list of effort multipliers of COCOMO in Section 2.3.1.2.

properties of the company and the development staff are considered during the estimation process, because project experience or historical project data is generated inhouse. However, a sophisticated development environment does not necessarily lead to good estimates. Since sophisticated development techniques raise productivity, effort estimates are presumably conformed to the achieved productivity level. In consequence, a project might be completed in 80% of the time that would be required by a less productive competitor. Nevertheless, this project might still have caused cost overruns of 30% according to inappropriate effort estimation.

According to the discussed nature of software development, building software is still a craftsmanship of skilled individuals, and, therefore, the individual developers finally determine the success of a project.[688] In the description of COCOMO'81 and COCOMO II, we have also seen that staff factors, like analyst and developer capability, have the highest impact on project performance.[689] Assuming that techniques like estimation by analogy and expert judgment automatically consider technological, product-specific as well as project-specific factors, the staff-oriented factors must still be taken into account.

With regard to the high impact of staff factors, if the individual differences are not well known, and, in consequence, they cannot be precisely incorporated in the effort estimation, the actual efforts of the project will show a high variance, which negatively affects the estimation accuracy. In view of this, it is *not* surprising that expertise-based estimation techniques can result in high estimation accuracy.[690] Experts can intuitively take account of the *human aspects* of a development project. Senior project managers most probably know who is skilled and reliable, who is good in prototyping, and who is best in finishing projects and assuring high quality work. At the same time, they know who might procrastinate, who fills available time with gold-plating, and who needs motivation and coaching in order to work on the project goals. The explorative analysis of the *Exercise Dataset* has given evidence that effort variability caused by individual differences results in precise estimates, when assuming a simple form of project control and a high intensity of the PARKINSON effect.

688 See Section 3.2.

689 These factors were gained by regression analyses based on empirical project data. See Section 2.3.1.2, 2.3.1.3, as well as 2.3.1.4.

690 See Section 2.4.

## 5.1.4 Building Block 4: Accidental Complexity and Invisibility of Software

In his book *"The Mythical Man-Month,"* published in 1975, FREDRICK BROOKS describes four inherent properties of software systems: *complexity, conformity, changeability,* and *invisibility*.[691]

### 5.1.4.1 Complexity

BROOKS argues that software – in terms of size – is more *complex* than any other human construct. According to the fundamental ideas of high-level programming languages[692], no two parts of a software system will look alike. Any piece of code that is at least needed twice becomes a *unique* subroutine in a way that it can be efficiently *reused*. For BROOKS, this alone is enough to differentiate software from tangible products like computer hardware or buildings as their production is dominated by repetition.[693]

BROOKS divides complexity into *essential* and *accidental* complexity. *Essential complexity* is problem inherent and thus unavoidable by definition, since any *simpler* solution would not solve the problem. In contrast, *accidental complexity* is avoidable since it refers to a complexity that we create on our own, i.e., it originates from the process or the techniques of solving the problem. BROOKS argues that:

> *"The complexity of software is an essential property, not an accidental one. Hence, descriptions of a software entity that abstract away its complexity often abstract away its essence. For three centuries, mathematics and the physical sciences made great strides by constructing simplified models of complex phenomena, deriving properties from the models, and verifying those properties by experiment. This paradigm worked because the complexities ignored in the models were not the essential properties of the phenomena. It does not work when the complexities are the essence."*[694]

The inherent complexity of software makes communication among developers difficult, it causes low software quality, cost overruns, as well as schedule delays. Finally, software has a great demand for learning und understanding, which *"makes personnel turnovers a disaster."*[695]

[691] BROOKS (1975).
[692] WILKES *et al.* (1952).
[693] BROOKS (1995b).
[694] BROOKS (1987), p. 11.
[695] BROOKS (1995b), p. 184.

Accidental complexity varies with developers' expertise since a problem can be solved in different ways. Different solutions, however, might require different levels of knowledge and experience. BIGGELEBEN demonstrates the general presence of accidental complexity when working with class libraries of common programming languages.[696] His work promotes the construction and utilization of frameworks in order to focus on the essence by shortening the gap between requirements and their implementation (see Fig. 5.7).

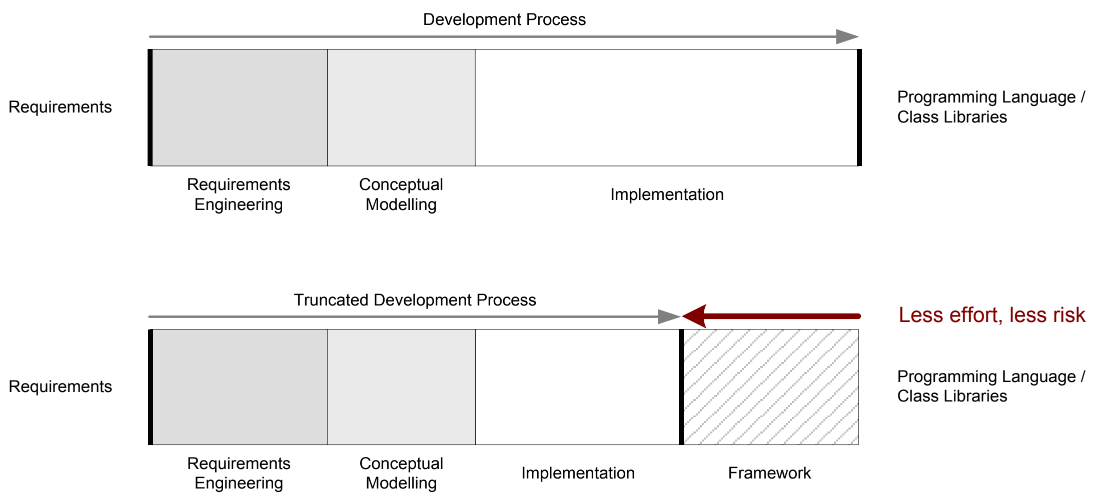


**Fig. 5.7:** Shortening the gap between requirements and their implementation[697]

Although the usage of frameworks requires a certain amount of training, their benefit is reducing effort and risk during the implementation phase. Especially, in the context of web-based development, frameworks like *Prototype*, *Scriptaculous*, and *JQuery* are well-received among developers.[698] Another (web application) framework that recently aroused much attention is *Ruby on Rails* which stands out due to its unique incorporation of programming principles like *Convention over Configuration* as well as *DRY ("Don't Repeat Yourself")*.[699] Basically, all of these frameworks offer frequently required functions that are pre-built, easy to use, and well tested across different browser platforms. In consequence, such frameworks reduce accidental development effort as well as the risk of unnecessarily spending time with bug-fixing.

### 5.1.4.2 Conformity

In contrast to fields like physics or chemistry, where researchers identify and work with *unifying* principles, the inner structure of software systems appears arbitrary:

[696] BIGGELEBEN (2007).
[697] See BIGGELEBEN (2007).
[698] FUCHS (2009); PROTOTYPE CORE TEAM (2009); RESIG (2009).
[699] HEINEMEIER HANSSON (2008).

> *"Much of the complexity he [the software developer] must master is arbitrary complexity, forced without rhyme or reason by the many human institutions and systems to which his interfaces must conform."*[700]

BROOKS argues that much complexity comes from the need to conform software systems to other interfaces. This complexity does not result from design flaws. Instead, complexity based on the demand for conformity is rather natural and cannot be removed by any redesign of the system.[701]

#### 5.1.4.3 Changeability

Software systems are permanently subject to change – during their operational phase or even during development. Tangible products, like computer hardware, buildings, or cars can be subject to change as well. However, tangible, manufactured products are changed much less frequently than software. BROOKS suggests that:

> *"Partly it is because software can be changed more easily – it is pure thought-stuff, infinitely malleable. Buildings do in fact get changed, but the high costs of change, understood by all, serve to dampen the whims of the changers."*[702]

The need for change depends on how software is used and perceived. At first, successful, widely accepted software systems will be used at the edge or beyond the original, intended domain. Software might be used in a way it was not designed for.[703] It co-evolves with its application domain and, therefore, requires change.[704]

Second, successful software might outlive its technical environment. The technical infrastructure might become obsolete over time until it gets replaced by new technology. In this case, the software must be changed in order to conform to its new environment.[705]

#### 5.1.4.4 Invisibility

Software is immaterial and thus *invisible*. Moreover, software cannot be visualized. Illustrations, like Entity-Relationship Models or UML Class Diagrams – despite their

[700] BROOKS (1995b), p. 184.
[701] BROOKS (1995b).
[702] BROOKS (1995b), pp. 184-185.
[703] BROOKS (1995b).
[704] LEHMAN (1980); LEHMAN (1996). See Section 3.1.2.2.
[705] BROOKS (1995b).

usefulness – are only geometric *abstractions*.[706] Because software is neither two nor three dimensional, it has no natural spatial representation. BROOKS argues that:

> *"As soon as we attempt to diagram software structure, we find it to constitute not one, but several, general directed graphs superimposed one upon another. The several graphs may represent the flow of control, the flow of data, patterns of dependency [...]. Indeed, one of the ways of establishing conceptual control over such structure is to enforce link cutting until one or more of the graphs becomes hierarchical."*[707]

Visualizations of software systems are always based on simplification and the restriction to a particular, isolated perspective. Visualization techniques do not provide a general view on the system, which reflects all causalities and dependencies. As a result, the general absence of sufficient visualization techniques has a severe, negative impact on designing, communicating, as well as understanding software systems.[708]

#### 5.1.4.5 General Impact on Software Development

The presented, inherent properties of software systems explain the irreducible essence of software. On the question whether software development has to be hard or whether there is a *silver bullet* for attacking the essence, BROOKS writes:

> *"I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. [...] If this is true, building software will always be hard. There is inherently no silver bullet."*[709]

Complexity turns software development into a naturally ambitious and challenging task. Software developers are confronted with both the essential and the accidental complexity. Since accidental complexity originates from the process of problem solution, only the accidents can be reduced by technological progress, for example, by introducing new programming concepts, domain-specific languages, or application frameworks.

[706] BROOKS (1995b).
[707] BROOKS (1995b), pp. 185-186.
[708] BROOKS (1995b).
[709] BROOKS (1995b), p. 182.

BROOKS argues that even if 90% of the overall complexity were based on accidents, the elimination of all accidental complexity would still not give an order of magnitude improvement.[710] In a recent panel discussion on *"Hopes for the Silver,"* BROOKS adjusted the ratio of essential and accidental complexity to past technological improvements (see Fig. 5.8):

> *"The great leaps of progress in the past were accomplished by eliminating accidental difficulties [...]. Of the remaining difficulties, at least half seem to me to be essential, the very inherent complexity of what we build. Therefore, no attack on accidental difficulties can bring an order-of-magnitude improvement – indeed, more than a factor of 2."*[711]

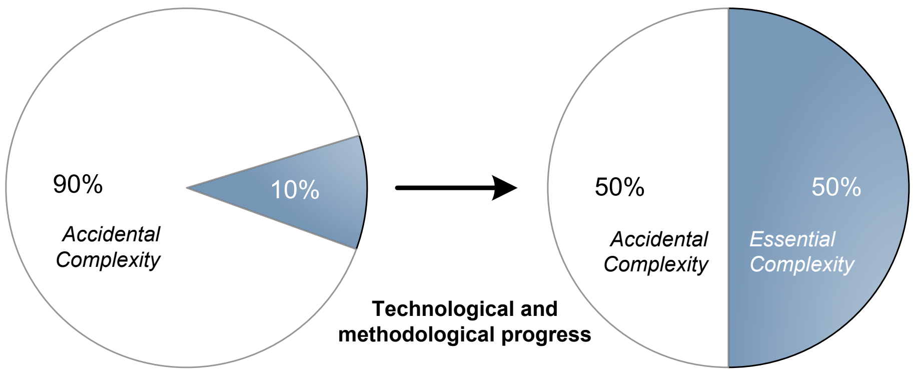


**Fig. 5.8:** Past and recent proportions of essential and accidental complexity

Regardless whether a ratio of 10:90 or 50:50 is more plausible today, these ratios show that the room for improvements based on eliminating accidental complexity is limited. As a result, software developers might soon – unless they already do – face a point of improvement, which does not provide major productivity gains any longer. They might get stuck between minor productivity gains and huge learning efforts that are required to master new technologies. However, the ratio of essential and accidental complexity does not exclusively determine the effort required by a particular developer. Individual differences across developers, for example, different levels of experience and capability, must also be considered when turning to development effort. Correspondingly, another member of the aforementioned panel discussion states that:

> *"Striving for excellence is the real silver bullet that will deliver an order-of-magnitude improvement through growth, both personal and professional. The silver bullet must come from within, rather*

[710] BROOKS (1995b).
[711] FRASER *et al.* (2007), p. 1027.

> *than without. We are the Silver Bullet—which we achieve by professional excellence."*[712]

### 5.1.4.6 Impact on the Causal Model

To reprise, accidental complexity refers to a complexity that we create on our own. It originates from the process or the techniques of solving the problem. The degree of complexity depends both on the particular problem as well as the particular developer. Due to individual differences, e.g., experience and capability, different developers will most probably favor *different* solution approaches when faced with *identical* problems.

Experienced and skilled developers might prefer approaches that make use of sophisticated frameworks, libraries, or components, which on the one hand require a significant amount of time to learn, but on the other hand decrease solution effort extensively.[713] Ergo, they can reduce accidental complexity as well as the risk of getting stuck with irrelevant details.

In opposition, novice developers try to solve a problem completely by hand because they are unaware of more sophisticated solution approaches. Besides facing a higher degree of accidental complexity, novice developers usually need more effort for writing code than experienced developers. They might also waste time with side effects and bugs more often.

For project lead, it is hard to observe whether a particular developer works on a difficult solution or whether he or she is not challenged and, therefore, able to complete the work package ahead of schedule instead. The invisibility of software makes tracking progress even more difficult.[714] Developers will always find ways to appear busy, which restricts project lead to tracking progress on or above the work package level.

In order to exert control, project lead must know about individual differences across developers. According to the explorative analysis, this appears to be the only way of reducing estimated effort without risking equivalent cost overruns caused by less experienced developers.[715]

712 FRASER, MANCL (2008), p. 92.
713 See Section 5.1.4.1.
714 See Section 5.1.4.4.
715 See Section 4.3.8.

### 5.1.5 Summary of Causal Relationships

With respect to comprehensibility, this subsection picks up the considerations discussed above and consolidates the supposed causal relationships of software development projects into one causal model. This model describes an organizational setting that is supposed to promote estimation fulfillment:

1) Software developers show individual differences in terms of capability, experience, talent, learning behavior, etc. Accordingly, developers perform differently during development projects.

2) Besides, developers gain experience and learn new skills over time. This process of learning and advancing is also different between developers. Some might advance quickly, while others already stagnate on a particular learning plateau.

3) In past projects, single work packages were completed on time, ahead of schedule, or behind schedule. Generally, work packages differ in terms of difficulty and size. Whether or not a work package was delivered on time, mainly depends on its difficulty and size as well as the individual skill set of the responsible developer.

4) Project lead gains experience over time and learns how to estimate future effort based on analogy or intuition. The most important instruments for effort estimation are breaking down the software system into smaller elements, breaking down the development project into work packages, as well as the identification of analogies.

5) Effort estimates are produced and communicated at early project stages. Estimates (unconsciously) become target values.

6) Project lead tracks progress according to the master project plan, which is based on the approved estimates. If a project gets behind schedule, project lead will take action in order to put the project back on track.

7) Estimates also become target values for developers. If developers require too much time for completing work packages, and exceed estimated effort, project lead will intervene. In opposition, if developers are generally able to finish work packages ahead of schedule, they have three options to conform to the estimates. First, develop-

ers can procrastinate (*Student Syndrome*). Second, they can fill the available time, for example, with gold-plating (PARKINSON). Third, they perceive the estimate as an essential part of work package specifications. According to the effort estimate, developers adapt their interpretation of the solution to implement. In consequence, two identical requirements specifications only differing in the estimated effort, lead to two different solutions.

8) In view of that, work packages that are completed on time or behind schedule might outnumber work packages that are completed ahead of schedule. Therefore, projects generally tend to exceed the estimated overall effort. Project lead, however, monitors overall effort and tries to minimize effort overruns.

9) As a result, projects exceed estimated effort by moderate percentages, i.e., 20-30%, on average. Except for rare outliers, projects do not end with effort underruns. Consequently, estimated effort and actual project effort are not two independent observations. A particular organizational setting leads to estimation fulfillment.

10) Finally, the individual differences between developers and their appropriate consideration during effort estimation are supposed to influence the average percentage of effort overruns. If individual differences are taken into account precisely, the deviation of effort estimate and actual project effort will have a lower spread.

## 5.2 Predictive Component

### 5.2.1 Central Theory Statement

The interplay between project lead and developers, the craftsmanship of software development, as well as the inherent properties of software, especially complexity and invisibility, give causal explanations for a particular organizational setting of software development projects. Based on this causal model, the central statement of the *Estimation Fulfillment Theory* reads as follows:

*The ex-ante estimation of software development effort and the ex-post observation of actual project effort are not independent. Especially, if effort is estimated by expertise-based estimation techniques, the effort estimate will become an essential part of the specification and turn into a self-fulfilling prophecy. Both project lead and developers have interests and means to fulfill the approved estimate. In view of that and due to the individual differences across developers, actual effort will exceed any plausible estimate.*

In general, the theory is *limited* to team- and project-based software development projects.

### 5.2.2 Assumptions

According to the supposed causality of software development projects, this theory acts on the following specific and testable assumptions:

***Assumption 1 (A1):*** *Practitioners favor expertise-based techniques, i.e., Work Breakdown Structures, Estimation by Analogy (or intuition), and/or (Wideband) Delphi.*

***Assumption 2 (A2):*** *Expertise-based techniques allow early estimates based on premature requirement specifications. Accordingly, if practitioners estimate effort by expertise-based techniques, estimates are produced and communicated at early project stages.*

***Assumption 3 (A3):*** *Project lead tracks progress and takes necessary actions in order to prevent the project from getting behind schedule. Thus, project lead conforms to the given estimates.*

***Assumption 4 (A4):*** *Developers tend to procrastinate, fill available time with gold-plating activities, and/or perceive effort estimates as an essential part of work descriptions. Estimates, therefore, significantly influence how developers interpret and process work packages. As a result, developers conform to effort estimates.*

***Assumption 5 (A5):*** *The productivity of developers significantly varies due to individual differences, e.g., different levels of capability, experience, and learning plateaus.*

Besides these five verifiable assumptions, two general assumptions are made. First, it is assumed that employees who are responsible for effort estimation have necessary skills to produce *plausible* estimates, i.e., estimates are not absurd and do not generally make the completion of projects impossible. Second, effort estimates are not generally affected by continuous overestimation.

If all assumptions are satisfied, the observation of actual project effort will not be independent of observed effort estimates. In consequence, the particular project or a whole organization will be affected by *estimation fulfillment* (EF). In the following, projects or organizations that are affected by estimation fulfillment are called *EF-type projects* or *EF-type organizations* respectively. Conversely, projects or organizations that are unaffected are called *I-type projects* or *I-type organizations* (independent estimation).

### 5.2.3 Propositions

Given that both project lead and developers conform to approved estimates, the consideration of the presented causal model leads to the following three propositions:

> ***Proposition 1 (P1):*** *Except for rare outliers, actual project efforts will not fall below estimated efforts if the project is an EF-type project.*

> ***Proposition 2 (P2):*** *Actual project efforts will exceed estimated efforts by a moderate percentage, i.e., 20%-30%, on average, if the project is an EF-type project.*

> ***Proposition 3 (P3):*** *If individual differences between developers are systematically and precisely documented and taken appropriately into account during effort estimation, the deviation of estimated and actual effort will reduce for EF-type projects.*

The first two propositions P1 and P2 will be subject to an empirical validation on the basis of expert interviews with software professionals. This approach, however, is not suited for validating proposition P3. Its validation requires a broader comparative analysis of EF-type companies, which differ in how they take individual differences into account. This analysis is not part of this thesis. Nevertheless, relevant interview responses concerning proposition P3 will be presented. Additionally, the rationale behind P3 will also be discussed during the expert review.

Before turning to the validation of the theory, the empirical observability of estimation fulfillment is discussed first. The following section introduces the idea of *estimation fingerprints*. Afterwards, a survey of closely related research is given.

## 5.3 Statistical Perspective

Depending on the estimation technique, the experience of staff members who estimate effort, the individual differences across the developer staff, as well as project-specific factors, software companies have specific distributions of estimated and actual effort over time. Given historical project data of one company, this specific distribution becomes *empirically observable*. Such distributions or density functions, respectively, can be interpreted as *estimation fingerprints* for specific types of effort estimation and its impact on actual project performance. In order to analyze and visualize the relationship of estimated and actual effort, a new variable is introduced which is based on the variable $z$ that has been defined in Section 4.3.5 (see Eq. 4.7):

$$z^{-1} = \frac{1}{z} = \frac{a}{e} \qquad \textbf{Eq. 5.2}$$

We use the reciprocal value of $z$ at this point, since it describes effort over- or underruns more intuitively. If $z^{-1}$ is 1.0, the estimated and actual efforts perfectly match. If $z^{-1}$ is greater than 1.0, the project caused cost overruns. A value of 1.5 or 2.0, for example, refers to cost overruns of 50% or 100% respectively. Conversely, if $z^{-1}$ is less than 1.0, the project was completed ahead of schedule.

In the following, different estimation fingerprints are introduced. A fingerprint refers to the individual shape of probability density functions based on $z^{-1}$. It has to be pointed out that such density functions do not assume a specific theoretical distribution like the normal, log-normal, or RAYLEIGH distribution. Yet, the supposed empirical distributions are all based on a bell-shape that might be skewed. In order to specify the presented fingerprints, general descriptive statistics like arithmetic mean, variance, skewness, and kurtosis are discussed.

### 5.3.1 Estimation Fingerprints

#### *Unbiased Estimation*

As a start, the following Fig. 5.9 describes a balanced, unbiased, and low spread distribution of $z^{-1}$.

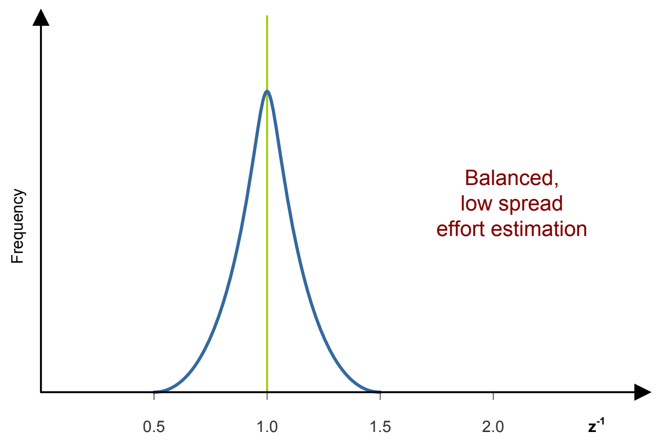


**Fig. 5.9:** Balanced, unbiased, low spread effort estimation

The mean of the given distribution is 1.0. Accordingly, most effort estimations are accurate, and correspond with actual effort, which indicates that the estimation is not *biased*. Moreover, the distribution is *balanced*, as it is symmetric. Therefore, effort overruns can be compensated with effort underruns in the long run. If the distribution were right-skewed, the skewness would indicate effects like PARKINSON or procrastination. Finally, this distribution has an acceptable peakedness, since most estimates stay within a 25% tolerance (*Pred(25)*) according to the mean.

This fingerprint describes the ideal type of effort estimation. It is neither biased by optimism nor by self-fulfillment. A closely related unbiased fingerprint is illustrated in Fig. 5.10. This distribution has the same properties except for a higher spread, which indicates lower estimation accuracy.

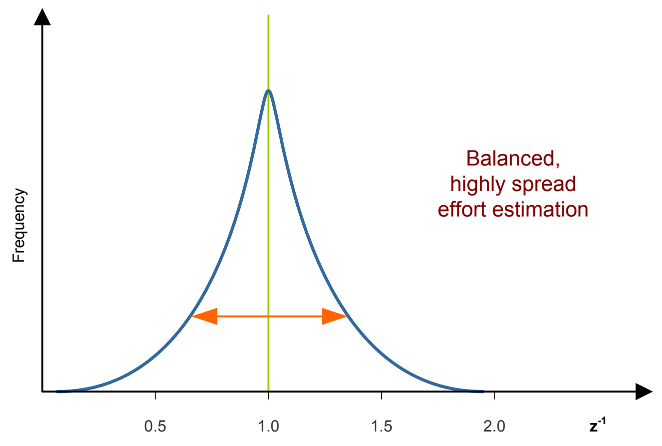


**Fig. 5.10:** Balanced, unbiased, highly spread effort estimation

### *Underestimation*

In contrast to the presented unbiased fingerprints, Fig. 5.11 shows an example of *biased* effort estimation. Generally, most of the actual efforts exceed the estimates.

Accordingly, the mean of $z^{-1}$ is above 1.0. This is an indication of either optimism bias or the continuous use of biased effort averages. The distribution, however, is still balanced, which indicates that neither PARKINSON nor procrastination causes this bias.

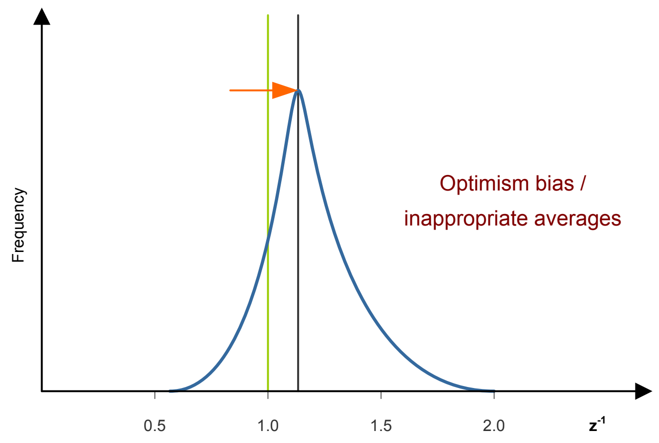


**Fig. 5.11:** Biased effort estimation

### *Continuous Overestimation*

Analogously, Fig. 5.12 visualizes effort estimation that is biased by *continuous over-estimation*. For this type, the mean of $z^{-1}$ is less than 1.0. Again, this distribution is balanced, and only affected by pessimism.

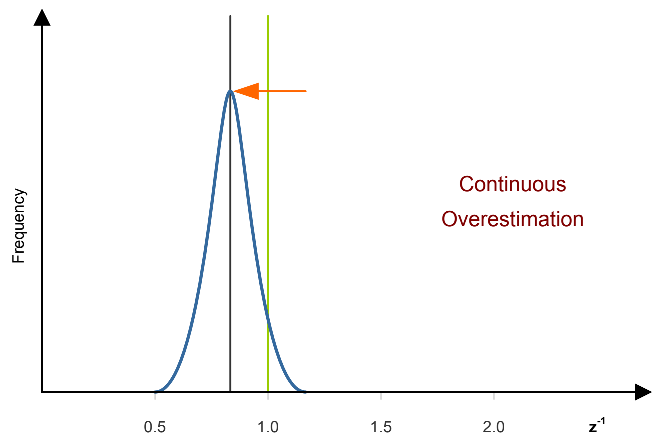


**Fig. 5.12:** Biased effort estimation due to continuous overestimation

### *Pseudo-perfect Estimation*

Continuous overestimation is not necessarily obvious as it can be hidden by the PARKINSON effect. In this case, quantitative analyses of estimates and actual effort can mistakenly lead to perfect estimation accuracy. On the one hand, overestimation can absorb escalated work packages. On the other hand, assuming that the PARKINSON effect applies with a high intensity, developers will generally fill the available time, for example, with gold-plating activities. As a result, nearly all work packages are completed on time. Fig. 5.13 shows the corresponding distribution of $z^{-1}$. Its high peakedness and low spread suggest perfect effort estimation. If continuous overestimation and PARKINSON affect the effort estimation, however, the fingerprint given in Fig. 5.13 does not result from high estimation accuracy but from a strong bias.

Extreme forms of overestimation would even turn $z^{-1}$ into a left-skewed distribution assuming that the PARKINSON effect cannot consume all buffer time (denoted by the dashed left-skewed distribution in Fig. 5.13). In this case, lots of projects are completed with less effort than estimated without having as many counterparts on the effort overrun side. As this asymmetry reduces the mean of $z^{-1}$ below 1.0, this alternative fingerprint lies between *Continuous Overestimation* and *Pseudo-Perfect Estimation*.

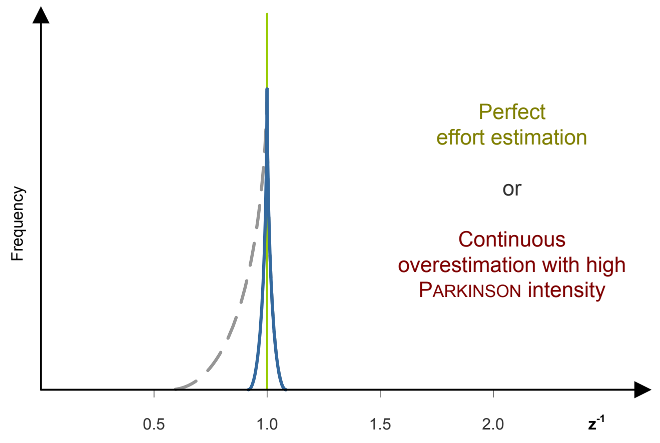


**Fig. 5.13:** Pseudo-perfect effort estimation

### *PARKINSON-affected Estimation*

Generally, procrastination and PARKINSON both affect $z^{-1}$ in a way that it becomes right-skewed. In this case, effort overruns cannot be compensated by effort underruns. The estimates are therefore unbalanced and biased. The observation that no project is completed with effort underruns gives strong evidence for the PARKINSON effect. *Parkinson-affected Estimation* can be distinguished from *Underestimation* by looking at the skewness of $z^{-1}$. *Underestimation* caused by a permanent optimism bias, for example, is symmetric while *Parkinson-affected Estimation* shows a typical right-skewed shape. The corresponding distribution of $z^{-1}$ is given in the following Fig. 5.14.

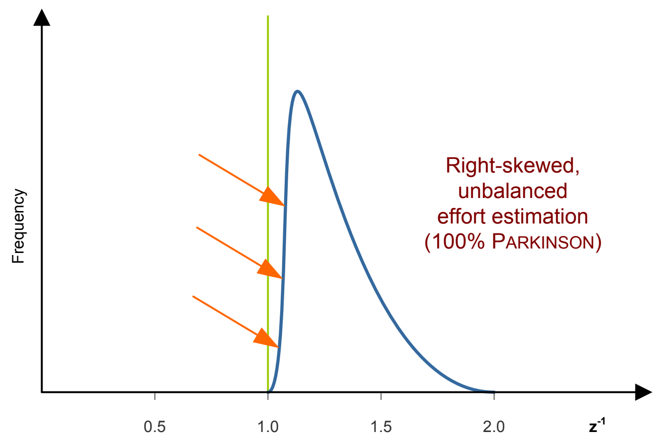


**Fig. 5.14:** Right-skewed, unbalanced effort estimation

### *Self-fulfilling Estimation*

Finally, the last fingerprint describes self-fulfilling effort estimation. This fingerprint has three important characteristics. First, nearly all actual efforts are above the estimates. Thus, all values of $z^{-1}$ are greater than 1.0. Projects completed with effort underruns are the exception. Second, the distribution of $z^{-1}$ is fairly symmetric. Third, the mean of $z^{-1}$ refers to an acceptable average effort overrun of, for example, 20% (1.20). Fig. 5.15 visualizes an exemplary distribution of $z^{-1}$ based on self-fulfilling effort estimates. If a company has this particular fingerprint, it can be assumed that the interests of project lead and developers are set in such a way that projects generally exceed estimated effort. This bias represents a quantitative evidence for estimation fulfillment.

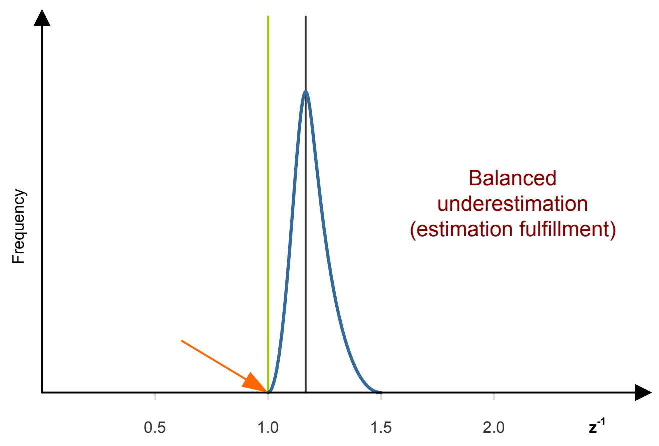


**Fig. 5.15:** Estimation Fulfillment

## 5.3.2 Fingerprint Classification

In order to evaluate which fingerprint matches the effort estimation of a company, historical project data must be available. Generally, the more project data is available, the more significant are accuracy analysis results. When assessing a company's effort estimation by analyzing the distribution of $z^{-1}$, practitioners as well as researches must focus on the *mean*, the *skewness*, as well as the *minimum* of $z^{-1}$. The following Fig. 5.16 gives a simplified classification tree that allows identifying which fingerprint is given. This classification tree is not complete, but it accounts for the characteristics of the discussed fingerprints.

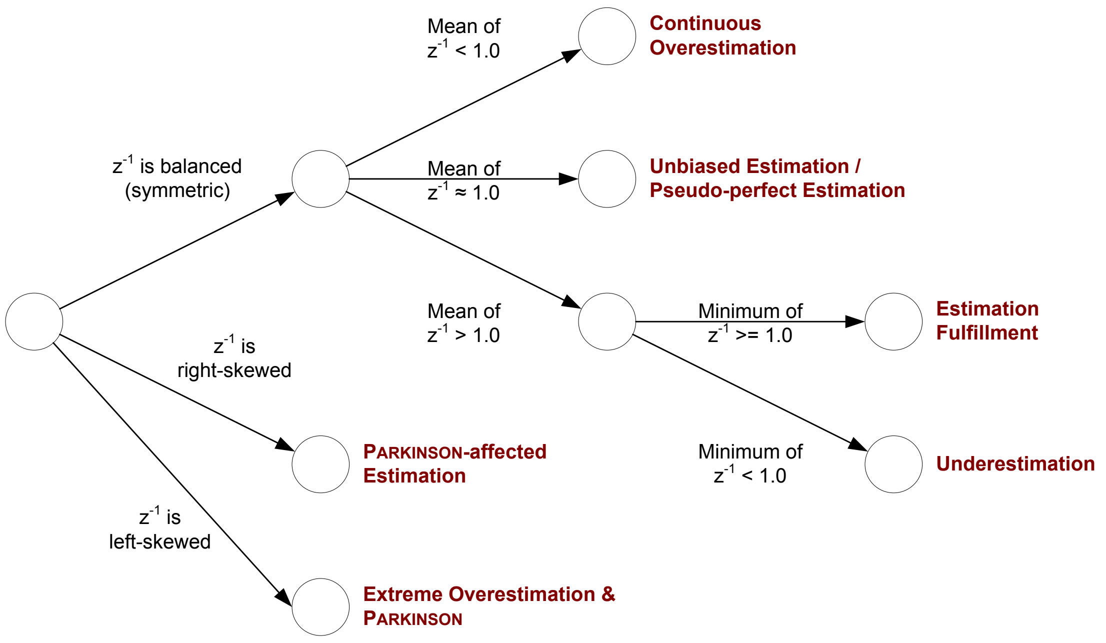

Mean of $z^{-1} < 1.0$
Continuous Overestimation
$z^{-1}$ is balanced (symmetric)
Mean of $z^{-1} \approx 1.0$
Unbiased Estimation / Pseudo-perfect Estimation
Mean of $z^{-1} > 1.0$
Minimum of $z^{-1}$ >= 1.0
Estimation Fulfillment
$z^{-1}$ is right-skewed
PARKINSON-affected Estimation
Minimum of $z^{-1} < 1.0$
Underestimation
$z^{-1}$ is left-skewed
Extreme Overestimation & PARKINSON

**Fig. 5.16:** Simplified classification tree for estimation fingerprints

## 5.4 Survey of Closely Related Research

The phenomenon of estimation fulfillment has not been subject to intensive research in the past.[716] There is only little research that addressed this issue. Generally, the possibility that estimates might become self-fulfilling is obvious. In textbooks on effort estimation, the self-fulfillment of estimates, if at all, is only mentioned as a side note. BOEHM, for example, argues:

> *"If a software development cost estimate is within 20% of the 'ideal' cost estimate for the job, a good manager can turn it into a self-fulfilling prophecy. The degrees of freedom, which allow the project manager to do this, are the common slack components of the software person's typical workweek. [...] These slack components*

[716] For example, the phrase *"estimation fulfillment"* does not lead to any appropriate results when searched via *Google Web Search* (six results in total). Likewise, *Google Scholar* does not find any articles. The same is true for related phrases, e.g., *"estimate fulfillment,"* or similar spellings. In literature, the phenomenon of interest is usually addressed by using the terms *"self-fulfillment"* or *"self-fulfilling estimates."*

> *– training, personal activities, general professional activities – comprise about 30% of the software person's time on the job."*[717]

Additionally, BOEHM discusses the PARKINSON effect as well as the deadline effect as further influences on project performance.[718] However, BOEHM'S brief argumentation (less than two pages of text) is restricted to the project management perspective, and his quintessence is that there is a synergy between estimates and good project planning and control, which allows managers to fulfill estimates.[719] With reference to GREGOR'S theory classification[720], BOEHM outlines drivers and means to conform to estimates, and, hence, he briefly addresses the questions *"what is"* as well as *"why."* Yet, his short argumentation neither explains *"when"* this happens (assumptions to be supported), what exactly *"will be"* (propositions), nor does it provide a detailed causal model.

Similarly, MCCONNELL writes about the purpose of estimates:

> *"Project planners often find a gap between a project's business targets and its estimated schedule and cost. If the gap is small, the planner might be able to control the project to a successful conclusion by preparing extra carefully or by squeezing the project's schedule, budget, or feature set. If the gap is large, the project's targets must be reconsidered. The primary purpose of software estimation is not to predict a project's outcome; it is to determine whether a project's targets are realistic enough to allow the project to be controlled to meet them."*[721]

MCCONNELL affirms the possibility of self-fulfilling estimates. Even more, he argues that the purpose of estimates is not just prediction but setting targets to which developers and managers commit. Elsewhere in his work, MCCONNELL also introduces the PARKINSON effect as well as the *Student Syndrome* as important influences on project performance.[722] However, those concepts and thoughts are not integrated in order to present a complete causal model that addresses the relationship of estimates and actual project efforts. Therefore, the questions *"what is"* and *"why"* are only addressed superficially. Moreover, the relevant building blocks are rather scattered throughout the entire work. Yet, MCCONNELL shortly focuses on and agrees upon the problem of permanent effort overruns:

[717] BOEHM (1981), p. 591.
[718] BOEHM (1981), pp. 592-593.
[719] BOEHM (1981), p. 594.
[720] GREGOR (2006), p. 620. See Section 1.3.
[721] MCCONNELL (2006), p. 13.
[722] MCCONNELL (2006), pp. 21-22.

> *"We often speak of the software industry's estimation problem as though it were a neutral estimation problem – that is, sometimes we overestimate, sometimes we underestimate, and we just can't get our estimates right. But the software [industry] does not have a neutral estimation problem. The industry data shows clearly that the software industry has an underestimation problem."*[723]

SOMMERVILLE also briefly addresses the idea of self-fulfilling estimates. Besides, he notices a lack of rigorous research on this issue:

> *"Project cost estimates are often self-fulfilling. The estimate is used to define the project budget, and the product is adjusted so that the budget figure is realized. I do not know of any controlled experiments with a project costing where the estimated costs were not used to bias the experiment. A controlled experiment would not reveal the cost estimate to the project manager. The actual costs would then be compared with the estimated project costs. However, such an experiment is probably impossible because of the high costs involved and the number of variables that cannot be controlled."*[724]

However, the idea of dependence between estimates and actual project effort is only brought up in one sentence. The questions *"what is"* and *"why"* are not discussed. A predictive component (*"when"* and *"what will be"*) is not provided either. Nevertheless, SOMMERVILLE notices a lack of research, i.e., controlled experiments, as well as problems of conducting such experiments in this context.

JORGENSEN and SJOBERG address this lack of research by conducting case studies as well as controlled experiments with computer science students to analyze the impact of effort estimates on software project work.[725] However, they faced some difficulties when conducting the experiments:

> *"The low number of participants makes the interpretation of the p-values difficult. We interpret the results as a weak indication of a relationship between too low effort estimates and a pressure on the software work that, for small programming tasks, can lead to less use of effort on the cost of more errors. Clearly, larger, more realistic studies with professional software developers are needed to validate the size and significance of that relationship."*[726]

[723] MCCONNELL (2006), p. 27.
[724] SOMMERVILLE (2006), p. 620.
[725] JORGENSEN, SJOBERG (2000); JORGENSEN, SJOBERG (2001).
[726] JORGENSEN, SJOBERG (2001), p. 950.

Nevertheless, based on the findings of the case study analysis and the experiments, JORGENSEN and SJOBERG conclude:

> *"Pre-planning effort estimates can have a major impact on the detailed planning effort estimate. One of the experiments indicates that this impact is present even when the estimators are told that the early estimates are not based on historical data or expert knowledge. The awareness of the size of this impact seems low. [...] Effort estimates can have a strong impact on the project work. The size and nature of this impact depend, among other variables, on project priorities and completeness of requirements specification. For example, we found that projects with high priority on costs and incomplete requirements specifications were prone to adjust the work to fit the estimate when the estimates were too optimistic, while too optimistic estimates led to effort overruns for projects with high priority on quality and well specified requirements."*[727]

Another confirmation of estimates influencing actual effort is given by ABDEL-HAMID and MADNICK.[728] In the description of their integrated *System Dynamics* approach, they argue:

> *"The critical point is that a different estimate creates a different project. This phenomenon is somewhat analogous to the 'General Heisenberg' principle in experimentation: 'When experimenting with a system about which we are trying to obtain knowledge, we create a new system'. [...] By imposing different estimates on a software project we create different projects."*[729]

In order to model and simulate the impact of estimates on actual project performance, the system dynamics model contains a *recursive flow*, including the project progress, the project status, man-day shortage, communication overhead, as well as workforce hiring and firing (see Fig. 5.17). Why these elements were selected, is only reasoned in a few words.[730] Yet, this issue has been addressed by ABDEL-HAMID and MADNICK in a previous publication on the impact of schedule estimation on software project behavior.[731]

[727] JORGENSEN, SJOBERG (2001), p. 951.
[728] ABDEL-HAMID, MADNICK (1991).
[729] ABDEL-HAMID, MADNICK (1991), p. 169.
[730] ABDEL-HAMID, MADNICK (1991), p. 170.
[731] ABDEL-HAMID, MADNICK (1986).

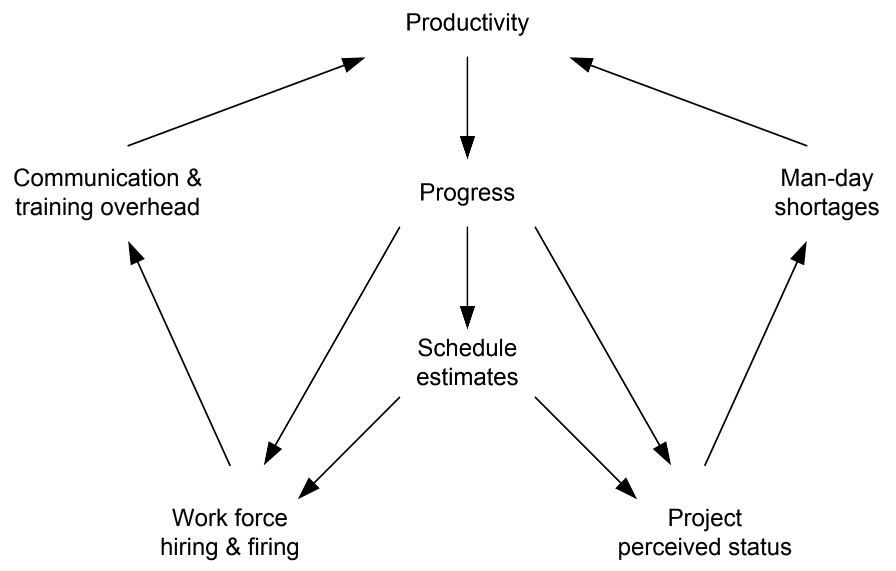


**Fig. 5.17:** The feedback impact of schedule estimates as suggested by ABDEL-HAMID and MADNICK[732]

Although the focus of ABDEL-HAMID and MADNICK's work on system dynamics is predominantly on predicting *"what will be,"* they have given comprehensive explanations on *"what is"* and *"why"* in the aforementioned previous publication. However, their work can be understood rather as an instrument than as a theory or specific research results. Correspondingly, they explain:

> *"The objective of this [research] is to enhance systematically our understanding of and gain insight into the general process by which software development is managed. [...] The [developed integrative system dynamics] model was used as an experimentation vehicle to study or predict the dynamic implications of an array of managerial policies and procedures."*[733]

In comparison to other research efforts, the work of ABDEL-HAMID and MADNICK has the most similarities with the Estimation Fulfillment Theory. Accordingly, it must be considered as the most closely related research. However, besides numerous parallels with the Estimation Fulfillment theory, there are major differences:

1) The research approaches, i.e., EP theory construction vs. developing a system dynamics model, are completely different.

2) The research objectives are different. While the Estimation Fulfillment Theory exclusively concentrates on the dependence of esti-

[732] ABDEL-HAMID, MADNICK (1986), p. 71.
[733] ABDEL-HAMID, MADNICK (1991), Preface.

mated and actual project effort, the System Dynamics approach addresses the entire software development process.

3) The major focus of the system dynamics model is on *schedule*, not on *effort*.

4) The system dynamics model does not address the individual developer. Therefore, the questions why individual developers conform to estimates, how estimates influence the developer's perception of specifications and work descriptions, what individual differences between developers imply, and by what means project lead reacts to delays, are either not discussed or argued in a different way.

5) ABDEL-HAMID and MADNICK do not provide limitations that explicate *"when"* estimates influence software project behavior and *"when"* they do not.

6) Their work is based on partially different assumptions, since they assume that

   *"Once requirements are fully specified and the architectural design phase is initiated, there will be no significant subsequent changes in the users' requirements."*[734]

Finally, it has to be pointed out that – despite obvious similarities – the specific answers and arguments given by ABDEL-HAMID and MADNICK on *"what is, why, how, and what will be"* are different to the EFT. Moreover, the forms of presentation are worlds apart. The EFT is presented as an EP theory with a clear causal model (*"what is, why, how"*), a set of assumptions (*"when"*), as well as a set of propositions (*"what will be"*). The comprehensibility of its presentation was double-checked by the expert review. In opposition, one must have profound knowledge on system dynamics modeling in order to understand and comprehend the arguments and findings of ABDEL-HAMID and MADNICK. Accordingly, it is questionable whether they successfully reach the relevant audience, which should include practitioners.

[734] ABDEL-HAMID, MADNICK (1991), p. 20.

# 6 TEST OF ESTIMATION FULFILLMENT THEORY

The *Estimation Fulfillment Theory* (EFT) is newly constructed, and, therefore, must be confronted with empirical content in order to get validated. This validation either results in confuting or corroborating the theory. As a first step, the assumptions and propositions were tested by a set of qualitative, semi-structured *expert interviews*. Second, the theory, the underlying causal model, the assumptions, and propositions were discussed with an expert in order to *review* the drawn conclusions and made assumptions during theory construction from the practitioner's perspective.

## 6.1 Research Method and Research Design

### 6.1.1 Design of Expert Interviews

Qualitative interviews are typically *"structured interviews," "unstructured or semi-structured interviews"* or *"group interviews."*[735] A group interview, however, is also either structured or semi-structured. Structured interviews are based on a complete script that leaves no room for improvisation. This type is often used in surveys, which do not require the presence of the researcher. In opposition, semi-structured interviews are based on an incomplete script. A set of questions is prepared beforehand, leaving room for improvisation during the interview.[736]

MYERS and NEWMAN list numerous difficulties, problems, and pitfalls in using qualitative interviews.[737] For example, interviews are often conducted with a *lack of trust* and a *lack of time*. Additionally, the *level of entry*, e.g., whether the interviewee is a junior or senior manager, might influence the interview result. Other noted problems

---

735 FONTANA, FREY (2000); MYERS, NEWMAN (2007).
736 FONTANA, FREY (2000).
737 MYERS, NEWMAN (2007).

are the *construction of knowledge* during the interview situation, the *ambiguity of language*, as well as the possibility that interviews can completely *go wrong*.[738]

In total, seven expert interviews were conducted. All interviews were semi-structured and based on identical scripts. Two interviews were group interviews with two interviewees. The interview script can be found in Appendix A.3. All interviews were conducted in the first half of 2009, in face-to-face meetings, and usually at the interviewee's company.

During the interviews, handwritten notes, including quotes, were taken and typed afterwards. Since the interviewees have to disclose business secrets, handwritten notes were preferred to a tape recorder. While some interviewees explicitly refused to be tape recorded, it was supposed that tape recording has a negative effect on the willingness to disclose sensitive information. Similarly, HAYES and WALSHAM argue: *"Detailed field notes were preferred to the use of a tape recorder, as it was thought that tape recording would have led to less candid responses."*[739] In general, the interview questions were designed with respect to the aforementioned problems and pitfalls.

Concerning the potential lack of trust, all interviewees were informed that the interviews can be done *anonymously*, and that they will receive a *One-way Non-Disclosure Agreement* (NDA). *"One-way"* means that only the researcher signs the contract, and only the researcher must ensure confidentiality. According to HAYES and WALSHAM, *"the initial part of the interview would be spent explaining the identity and purpose of the researcher(s), and reassuring interviewees that no attribution would be given to their views in any subsequent discussion or reports."*[740]

Additionally, all interviewees were informed about the estimated minimal duration of the interview. Pretests had shown that the interview could be completed in 1.5 hours if questions were only briefly answered. Since the actual duration strongly depends on the interviewee's willingness to explain and disclose internal matters, the interviewees could control the duration of the interview. They were encouraged to stop the interview if it took too long. Yet, no interview had to be stopped due to time problems or for other reasons.

All interview questions were kept as simple as possible and positioned in a deliberate order to avoid the construction of knowledge during the interview. A common ques-

[738] MYERS, NEWMAN (2007).
[739] HAYES, WALSHAM (2001), p. 268.
[740] HAYES, WALSHAM (2001), p. 268. MYERS, NEWMAN (2007), p. 22.

tion asked by the interviewees was *"What did other interviewees say?"* which was never answered during the interviews. However, after the official interview phase, interviewees who explicitly asked for it received feedback about their answers and a more detailed introduction on the actual purpose of this research. Before and during the interviews, the interviewees neither knew about the EFT nor were they informed that answers are partially used for theory testing.

The selection of interviewees originated from the personal network of the author (both private and professional). At first, based on direct contacts, a list of potential interview candidates was compiled. Each candidate was addressed via e-mail, phone, or personal meetings. They were asked about their involvement in effort estimation and the willingness to make an interview. Whenever possible, the heads of development were contacted as well. Afterwards, the interviewees were selected in terms of their position, responsibility, and years of working experience. Therefore, most interviewees have project lead experience, are senior managers and/or head of development. All of them are concerned with effort estimation.

All interviews have been conducted in German. The interview responses presented in the following were translated into English. Translated quotations are enclosed by double quotations marks.

### 6.1.2 Analysis of Expert Interviews

The expert interviews are used to *test* whether or not the assumptions and propositions of the EFT can be found in practice. Following POPPER, a theory can never be proven.[741] The empirical feedback of experts, however, can be used to *corroborate* or *falsify* the EFT.[742] Concerning the basic idea of *falsification*, POPPER summarizes:

> *"It should be noted that a positive decision can only temporarily support the theory [(corroboration)], for subsequent negative decisions may always overthrow it [(falsification)]."*[743]

In general, a theory must satisfy four requirements. These are *falsifiability*, *logical consistency*, *relative explanatory power*, and *survival*.[744] In order to test the survival of a theory, it must be confronted with empirical content. In this study, the experts' verbal responses are the empirical content the theory is confronted with. The *hypo-*

---

741 POPPER (1959).
742 LEE (1991).
743 POPPER (1959), p. 33.
744 POPPER (1959).

*thetico-deductive* model is applied to test *qualitatively* whether or not the interview responses corroborate or contradict the theory.[745] LEE explains:

> *"In hypothetico-deductive logic, the major premise is a general theory, the minor premise is a set of facts (the 'initial conditions') describing a situation, and the conclusion is what the theory predicts or hypothesizes to be observed in that specific situation. This means that, even if a theory is not directly verifiable because it refers to unobservable entities, it can still be tested indirectly, through the observable consequences (equivalently called 'predictions' or 'hypotheses') that are logically deducible from it. For example, the theory that 'All men are mortal' can be tested through its prediction that 'Socrates is mortal' by observing whether or not Socrates dies. This also means that no theory can be conclusively verified as true, since a new situation and a new prediction ('Plato is mortal') would reopen the possibility for its being disproven. A theory that is said to be 'confirmed' or 'corroborated' is one that has survived such a test, but remains open to being disproven in future tests."*[746]

Concerning *deductive* approaches as means of theory testing, LEE argues:

> *"In mathematical analysis, the validity of deductions involving mathematical propositions can be readily checked by turning to the rules of algebra. In qualitative analysis, there is no corresponding body of rules as succinct or easily applied as the rules of algebra for verifying the validity of deductions involving verbal propositions. To respond to this problem, it must first be emphasized that mathematics is a subset of formal logic, not vice versa. Logical deductions in the general case do not require mathematics. An MIS case study that performs its deductions with verbal propositions (i.e., qualitative analysis) therefore only deprives itself of the convenience of the rules of algebra; it does not deprive itself of the rules of formal logic, to which it may therefore still turn when carrying out the task of making controlled deductions."*[747]

Since the EFT consists of both a set of assumptions and a set of propositions, the assumptions must be tested first. If at least one of the assumptions is not satisfied, the theory does not apply at all, and, therefore, the corresponding interview can neither be used to corroborate nor to falsify the theory. In opposition, if all assumptions are satisfied, the propositions of the EFT must hold true (*corroboration*). If all assump-

[745] LEE (1991).
[746] LEE (1991), p. 345.
[747] LEE (1989), p. 40.

tions are satisfied, but at least one proposition fails in its application, the theory is falsified.

Accordingly, the analysis of an expert interview leads to one of *three* potential results that are described in the following.

*First*, if all assumptions are supported by an expert interview, it is supposed that the projects the interviewee refers to are affected by estimation fulfillment. If this is given, the two stated propositions (P1 and P2) must hold true, i.e., projects do not end with less actual effort then estimated and projects do have a moderate percentage of effort overruns on average (see Fig. 6.1). If the two propositions are supported, the interview results will *corroborate* the EFT.[748]

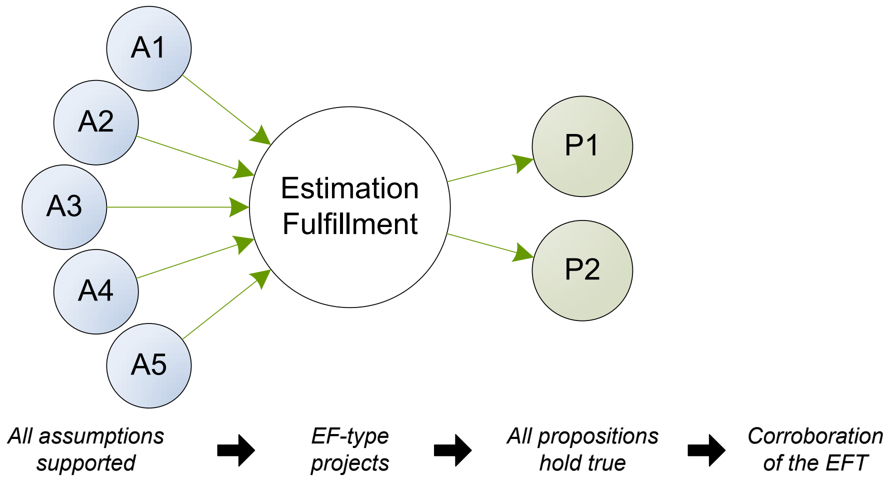


**Fig. 6.1:** Corroboration of the EFT

*Second*, if at least one assumption is unconfirmed, the interview does not allow further testing of the theory, since there is not enough evidence that the projects are affected by estimation fulfillment. In this case, the interview neither falsifies nor corroborates the EFT (see Fig. 6.2).

[748] POPPER (1959).

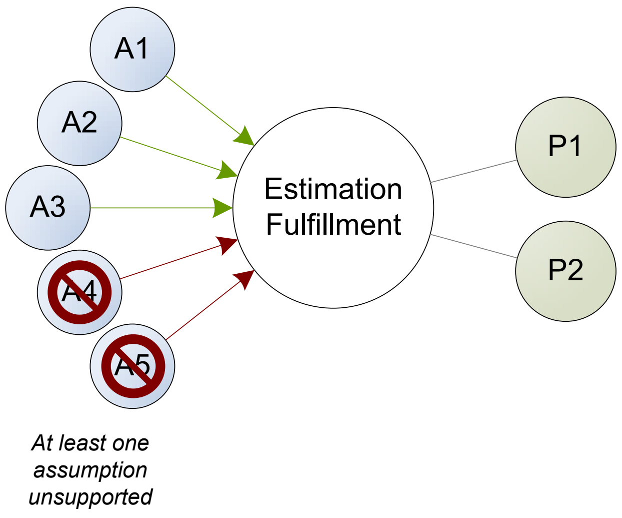


**Fig. 6.2:** Unsupported assumptions

*Third*, if all assumptions are supported, but at least one proposition does not hold true, the EFT in its particular form will be *falsified* (see Fig. 6.3).[749] However, *counter-findings* might also reveal differences between projects or companies that can be used to define further limitations of the EFT. If this is both feasible and plausible, the counter-findings do not necessarily falsify the theory, but narrow its applicability (*ad hoc adjustment*).[750] In other words, counter-findings might increase a theory's accuracy on the expense of relevance. Correspondingly, KUHN argues:

> *"All experiments can be challenged, either as to their relevance or their accuracy. All theories can be modified by a variety of ad hoc adjustments without ceasing to be, in their main lines, the same theories. It is important, furthermore, that this should be so, for it is often by challenging observations or adjusting theories that scientific knowledge grows."*[751]

[749] POPPER (1959).
[750] KUHN (1970).
[751] KUHN (1970), p. 13.

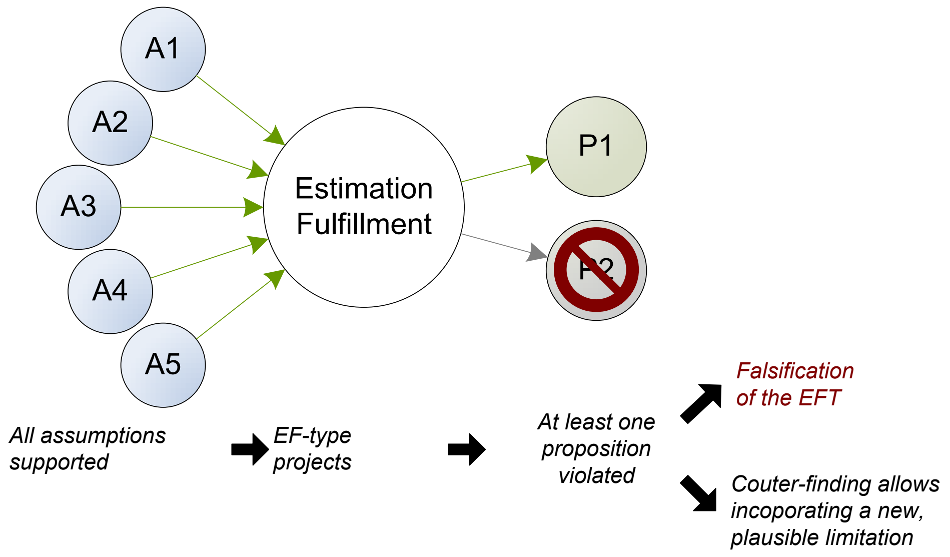

A1
A2
A3
A4
A5
Estimation Fulfillment
P1
P2
All assumptions supported
EF-type projects
At least one proposition violated
Falsification of the EFT
Couter-finding allows incoporating a new, plausible limitation

**Fig. 6.3:** Falsification of the EFT

For example, if one interviewee explained that his or her employee solely uses the Function Point method to estimate effort, one assumption (estimation by analogy) would be unsupported. In consequence, the interview would neither allow corroborating nor falsifying the theory. Conversely, if all assumptions were supported, but one interviewee reported that lots of projects end with effort underruns, Proposition 1 would be violated, and, consequently, the theory would be falsified.

Finally, the two general assumptions made, i.e. project managers can produce *plausible* estimates while not being affected by continuous overestimation, are not subject to an empirical validation in this research as they require an in-depth analysis of each interviewee, as well as each interviewee's company.[752]

## 6.2 Interview Results

In the following, the conducted expert interviews are briefly presented.[753] Besides the focus on the assumptions and propositions of the EFT, the biography of the interviewee, his or her professional experience, the employer, and especially the process of effort estimation are described. Since all interviewees wanted the interviews to be anonymized, the names of the interviewees and their employers were changed.

752 The two general assumptions are supposed to be satisfied due to economic considerations: If staff members who are responsible for effort estimation were generally unable to produce plausible estimates, they would get out of engagement in the log run. Similarly, if companies were affected by continuous overestimation, they would have severe problems in acquiring new projects.

753 A short overview of further lessons learned can be found in Appendix A.4.

### 6.2.1 Expert Interview #1

The first interview was conducted on January, 16th 2009 with BOB who is currently employed by *EnSwiss*, a Swiss company focusing on energy trading and supply. The interview took 4.5 hours. Biographical details of the interviewee are given in the following Tab. 6.1.

**Tab. 6.1:** Biographical details of BOB

| **Biographical details of the interviewee** | |
|---|---|
| Synonym | BOB |
| Year of birth | 1978 |
| Age | 31 |
| Nationality | German |
| Professional experience | 5.5 years |
| University / Graduation | Fachhochschule Münster, business administration, 2003 |
| **Current employment** | |
| Role | Analyst |
| Time of employment | 1/2008 until today |
| Job description | Requirements engineering, professional project lead, process design, documentation, Assistant head of development. |
| Completed projects | 4 |

For the last six years, BOB has been working for two different companies, which are both outlined in the following Tab. 6.2. BOB's department at EnSwiss is responsible for conducting large projects, which typically address the development, customization and/or introduction of *Energy Trading and Risk Management* (ETRM) as well as accounting systems.

**Tab. 6.2:** Employers of BOB

| **Company #1 "Energy One"** | |
|---|---|
| Synonym | Energy One |
| City, Country | Vienna, Austria |
| Turnover per year | 10-15 million € |
| Employees | 70 |
| Market | Software development, energy trading, consulting |
| Core competence | Energy Trading and Risk Management (ETRM) systems |
| Typical customers | Midsize and large energy companies |
| **Former Employment** | |
| Role | Product manager |
| Time of employment | 1/2004 until 12/2007 |
| Job description | Responsible for particular modules of the energy trading system. Requirements & Design. Project leadership. |
| Completed projects | 6 |

| **Company #2 "EnSwiss"** | |
|---|---|
| Synonym | EnSwiss |
| City, Country | Zurich, Switzerland |
| Turnover per year | > 1 billion € |
| Employees | > 500 |
| Market | Energy trading & supply. Focus of BOB'S department: Software development, project leadership |
| Core competence | Energy. BOB'S department: large projects, ETRM systems, accounting systems |
| Typical customers | BOB'S department: international subsidiaries, contractors of *EnSwiss* |
| Projects members | 8 – 40 |
| Project volumes | 500,000 to typically over 1 million € |

For his current employer *EnSwiss*, BOB is responsible for effort estimation. He describes the estimation process as follows:

> *We start each project with a kick-off workshop, where I meet two, three, or up to six representatives of the project owner. These representatives are usually experts. We discuss and collect business cases to get a picture of the software and the project goal. This kind of workshop lasts several days. The goal of the workshop is to identify the relevant business processes and to determine the boundaries, both of the software system and the project, in order to prevent a "moving target."*
>
> *After the workshop, I start writing a business concept, which takes one or two months. The business concept comprises all requirements of the software – that is, business requirements, technical restrictions, and further constraints. Both the customer and my supervisor receive a copy of the business copy for review.*
>
> *When the concept is completed, I meet with project lead. Based on the business concept, we create a full list of required tasks and a first, coarse-grained project plan. At this point, we produce the first effort estimates based on experience and analogies. The creation of the project plan usually takes some days.*
>
> *Next, I arrange an appointment with all relevant peer developers. This meeting is internal, so the project owner does not participate. My job is to present the business concept, explaining the business details, and informing about the project plan. I already present the effort estimates per work package. During this meeting, the peer developers give feedback and discuss the estimates with me. For complex tasks, they usually turn in their comments later. This meeting is repeated two or three times, each lasting about two hours. The result of this meeting series is a complete work breakdown for*

*the project including effort estimates and responsibilities per task. My boss gets a copy of this plan for review.*

*Afterwards, I meet with project lead in order to create the final, detailed project plan. At this point, we have received the reviews and change requests for the business concepts. For project planning, we simply use MS Project. Large projects can easily fill the walls of our corridors. The detailed plan contains all tasks, their dependencies, the responsible developers, as well as the planned durations.*

*The next step is the project kick-off, which is attended by project lead, the representatives of the project owner, peer developers, and – if required – external suppliers. We present the detailed project plan and inform about the schedule. When the project is started, we permanently monitor and evaluate the actual efforts per task in order to intervene if problems occur.*

Accordingly, effort estimation is based on an expertise-based technique, i.e., estimation by analogy and experience. Assumption A1 is therefore supported. Effort estimates are produced before project kick-off, and, hence, during an early planning phase. Moreover, BOB stated that actual efforts are permanently monitored and compared to the estimates in order to take actions. In view of that, assumption A2 and A3 are also supported. With respect to the PARKINSON effect, BOB answered:

*I do not adjust my working time consciously, but – when I think about that now – I do unconsciously. And when I continue thinking about that, this effect definitely applies for most – if not all – of my colleagues, too.*

Concerning the question whether or not he can work faster if deadlines have to be met, he stated that:

*Yes, but we also do overtime. I am not sure. Maybe I can increase my personal productivity by 20% by focusing on the critical task. Or I stay at home and work there. This definitely gives me a great increase of productivity since I will be less disturbed by daily business, for example, phone calls or discussions with colleagues.*

Accordingly, BOB and his colleagues tend to conform their working time to given estimates, which supports assumption A4. The question whether there is a noteworthy difference between developers in terms of capability and talent was clearly confirmed by BOB:

*"There are big differences. It's like day and night!"*

Moreover, he explained that the productivity ratio between developers is at least 1:4. He pointed out that there are also significant differences between developers in terms of domain knowledge and technical skills. Generally, if novice developers are employed, the productivity ratio between novice developers and experienced staff members is 1:5 or even more. Accordingly, assumption A5 is also supported.

Since all five assumptions are satisfied, the projects BOB participated in are supposed to be affected by estimation fulfillment. With regard to the deviation of estimated effort, BOB answered:

> *Effort overruns are around +20%. The worst case was +100%, which was a single case. However, we never have projects with effort underruns.*

This is in line with the propositions P1 and P2. In consequence, all assumptions and propositions are satisfied, and, therefore, the findings from this expert interview primarily corroborate the EFT.

Finally, with respect to proposition P3, BOB commented on the documentation and consideration of individual differences between developers:

> *Yes, we do after project completion. We conduct 'Lessons Learned' meetings with developers. Therefore, the result is rather qualitative. However, we use skill matrices with a four-point rating scale: 'Zero experience,' 'Basic Experience,' 'High Experience,' and 'Expert.' An entry can refer to anything, for example, general domain experience, energy trading, project lead experience, Java programming skills, or SQL.*
>
> *I think the skill matrices are very detailed and helpful. We cannot invest more effort in analyzing and documenting the skills of our developers without establishing something like an intelligence service.*

### 6.2.2 Expert Interview #2

The second interview was conducted on February, 5th 2009 with TOM who is employed by a large German software development and consulting company. The interview took three hours. Biographical details of the interviewee are given in the following Tab. 6.3. A brief summary of his current employer, *SolutionFactory*, follows in Tab. 6.4.

**Tab. 6.3:** Biographical details of TOM

| **Biographical details of the interviewee** | |
|---|---|
| Synonym | TOM |
| Year of birth | 1967 |
| Age | 41 |
| Nationality | German |
| Professional experience | 15 years |
| University / Graduation | Fachhochschule Wilhelmshaven, 2003 |
| **Current employment** | |
| Role | Solution Architect |
| Time of employment | 1997 until today |
| Job description | IT consulting & software development; business/technical requirements analysis; conception & design |
| Completed projects | 10-15 |

**Tab. 6.4:** Employer of TOM

| **Company Details** | |
|---|---|
| Synonym | SolutionFactory |
| City, Country | Frankfurt (and other cities), Germany |
| Turnover per year | > 250 million € |
| Employees | > 1.500 |
| Market | IT Consulting |
| Core competence | IT Consulting, Individual Solutions, IT Management, SAP Consulting, Application Management |
| Typical customers | Large German companies (finance, insurance, production, telecommunications) |
| Project members | 20-30 |
| Project volumes | 10-15 million € (last two projects) |

As a senior manager and solution architect, TOM is personally involved in the effort estimation process. Effort estimation at *SolutionFactory* was described as follows:

> *At first, we do a coarse-grained analysis of central and obvious requirements. Multiple staff members attend this process. Based on this analysis, we develop a primary solution idea. Next, this idea is decomposed into work packages. The effort per work package ranges from one day to one week. Now, we take an excel template and start to list all work packages. Each work package is briefly specified.*
>
> *The excel sheet is given to three colleagues, typically senior managers or experienced junior managers. For large projects, the sheet is given to five or six persons. Each recipient separately estimates effort for each work package. Effort is noted as a single value per work package. This process usually lasts one or two hours. Finally, all excel sheets are collected and the deviation between the*

> *estimates are subject to a group discussion. In this discussion, we adjust the estimates until everyone agrees.*

In view of that, the depicted effort estimation utilizes work-breakdowns, estimation by analogy and experience, as well as group discussions that are similar to Wideband Delphi. Assumption A1 is therefore supported. Moreover, the estimation process is based on early requirements and done before a detailed project plan is created. This satisfies assumption A2. Regarding assumption A3 *("project leads conforms to the given estimates")*, the interview did not lead to a clear discussion of this aspect. However, when explaining how developers react on deadlines, TOM explained:

> *I do not think that anyone can just work faster in order to meet a deadline. Developers rather refrain from unnecessary details or gold-plating respectively. Another way to meet deadlines is going for an 80% solution.*

If going for an 80% solution is interpreted as an action of project lead to conform to the project plan, assumption A3 will be supported. With respect to the PARKINSON effect, TOM explained:

> *I might fill available time if no new work packages are pending. Nevertheless, one might generally tend to do gold-plating. However, developers might not necessarily fill the entire available time. Maybe, they fill 50% of the time.*

The explanations of TOM concerning deadlines and PARKINSON lead to the conclusion that he and his colleagues tend to conform their working time to given estimates. Assumption A4 is supported. Concerning whether or not there are significant individual differences between developers, TOM stated:

> *Yes, these differences are extremely high. I think the variety of capability and talent across software developers is much bigger compared to other occupational groups. "And some developers are good for nothing."*

TOM assumed that his general productivity has at least doubled during the last 12 years of his professional experience. He continued:

> *According to my project experience, there are developers who need ten times the effort to complete work packages as I need – sad but true. There are also colleagues who have the same years of professional experience as me, but they have not advanced over the time. They are still on the level of an untalented junior developer. I think a good junior works at 30 to 50% of a good senior.*

According to his explanations, assumption A5 is satisfied. Therefore, the projects TOM took part in are supposed to be *EF-type* projects. TOM commented on effort overruns:

> *That's hard to say. As a general rule, projects exceed estimated effort. Since I am not personally involved with project reviews, I do not know precise percentages. However, I am quite sure that effort overruns are around 20 to 30%.*

With regard to effort underruns, he continued:

> *No, effort underruns were never observed for large projects. However, small projects have been completed ahead of schedule.*

As a result, proposition P1 is satisfied, but the previous explanation yields an interesting counter-finding regarding proposition P2. While proposition P2 is supported for *large* or *midsize* projects, there is a conflict with *small* projects. However, the project size is an obvious characteristic that allows differentiating projects. Thus, the project size can be used as a limitation in order to exclude small projects from the theory. In consequence, if small projects are excluded, proposition P2 is supported, which leads to a second corroboration of the EFT.

Finally, concerning proposition P3, TOM explained:

> *We have a skill information system, which is based on a job matrix. This matrix has 15 levels. A novice software developer usually starts at level 5, because you also find all secretaries and assistants in that system. The highest level ever given is 13. That's our Lead Architect. The matrix contains general capabilities, for example, project experience or project lead experience. Accordingly, there is no fine, detailed documentation of specific developer skills. Therefore, when staffing projects, we ask in a large circle "Who is good at what?"*

### 6.2.3 Expert Interview #3

The third expert interview was conducted on February, 13th 2009 with VINCE who is currently employed by *BankIT*, a German company focusing on software solutions and IT consulting for the financial sector. The interview took 4 hours. Biographical details of the interviewee are given in the following Tab. 6.5.

**Tab. 6.5:** Biographical details of VINCE

| **Biographical details of the interviewee** | |
|---|---|
| Synonym | VINCE |
| Year of birth | 1971 |
| Age | 38 |
| Nationality | German |
| Professional experience | 16 years |
| University / graduation | WWU Münster, Information Systems, 1998 |
| **Current employment** | |
| Role | Quality Manager |
| Time of employment | 1998 until today |
| Job description | Definition of development process models, monitoring development processes during projects, documentation & review, managing improvements, official release of software solutions, final inspection |
| Completed projects | 15-20 per year |

VINCE has been employed by *BankIT* for 16 years. His major responsibility is *quality management*. Therefore, he is concerned with most software development projects that are started by *BankIT*. His employer is briefly outlined in Tab. 6.6.

**Tab. 6.6:** Employer of VINCE

| **Company Details** | |
|---|---|
| Synonym | *BankIT* |
| City, Country | Undisclosed, Germany |
| Turnover per year | ~ 15 million € |
| Employees | ~ 130 |
| Market | IT Consulting, Software development |
| Core competence | Controlling systems, Risk controlling |
| Typical customers | Large and mid-sized European banks |
| Project members | 2-15 |
| Project sizes | 100-1,000 man-days |

With respect to the process of effort estimation, VINCE explained:

> *At first, we collect and list all major requirements. This list contains anything, for example, algorithms, GUI descriptions, or use cases. One staff member then produces the effort estimates according to this list. Sometimes more persons are assigned to effort estimation. All estimates are based on comparisons with and experience from past projects. The effort is estimated for creating the business requirements as well as design and implementation. Other aspects, for example, project management and testing, are estimated by surcharges. After finalizing the list, the result is discussed with a senior manager in order to challenge and adjust the estimates. Afterwards, if the required budget is approved, the project plan will be created.*

*Generally, a risk buffer of 15% is reserved for each project. This buffer, however, must be officially requested from upper management.*

Again, effort estimation is based on an expertise-based approach, which satisfies assumption A1. Since estimates are produced before the conception of the project plan, and, therefore, before developers start with design and implementation, assumption A2 is also supported. With regard to assumption A3, VINCE explained during the interview:

*We use Scrum. Accordingly, we arrange daily meetings with all developers for discussing critical problems as well as the general project status. These are very short meetings lasting 15 or 20 minutes. However, daily Scrums are important so that everyone stays informed and that project lead sees how the project evolves.*

Therefore, project lead at *BankIT* tracks progress. During the interview, the interviewee did not explicitly describe potential actions of project lead to respond to inconsistencies with the project plan. However, it is reasonable to suppose that daily meetings between project lead and developers entail reactions on deviations of a project. In consequence, assumption A3 is supported.

With respect to PARKINSON and deadlines, VINCE noted:

*For me, the Parkinson effect does not apply. However, the effect applies for developers. They fill available time with unnecessary activities. [...]*

*I think I can work faster in order to meet deadlines when switching into a "do not disturb" mode.*

Accordingly, developers conform to given estimates, which satisfies assumption A4. Concerning the individual differences between developers in terms of capability and talent, VINCE commented:

*Yes, there are perceptible differences. [...] There are developers who busy themselves with something for weeks, "but the result is nothing but trash." On the other hand, "if you assign two really good developers, they will always do well, no matter whether specifications were good or bad."*

This comment supports assumption A5. In consequence, the interview results support all five assumptions, which leads to the conclusion that estimation fulfillment can be found at *BankIT*. The deviation of estimated effort and actual project efforts was described as follows:

> *All projects usually consume the reserved risk buffer of 15 or 20%. Additionally, project effort still exceeds these buffers. Rarely, projects end with cost overruns of 40%. However, 40% refers to budget overruns, not effort. Generally, projects never show effort underruns. Maybe I have seen three projects in the last ten years with the actual effort below estimated effort.*
>
> *I must point out that effort and budget are quite mixed. Overtime and weekend working, for example, are not taken into account since developers do this on a "voluntary" basis.*

Accordingly, propositions P1 and P2 could be validated in this case. Correspondingly, the third expert interview also corroborated the EFT. Finally, with respect to proposition P3, VINCE explained:

> *We do not document developer capabilities systematically. However, we discuss target agreements with developers. Therefore, we select developers rather by intuition. Our developer pool contains 30 to 35 developers. During effort estimation, we do not know who will be assigned to a work package. We assume that this will be done by someone who's capable.*

### 6.2.4 Expert Interview #4

The forth expert interview, conducted on March, 18th 2009, was a group interview with PETE and MARC who are both employed by *Objects Inc*. This German company offers IT services to energy providers and public transportation services. *Objects Inc.* offers energy suppliers to outsource all customer-related processes, e.g., billing and settlement. PETE has been employed by *Objects Inc.* since its foundation in 1999. He works as a consultant for software solutions and has both project lead and effort estimation experience. MARC has been employed as head of software solutions at *Objects Inc.* since 2005. Biographical details of both interviewees are given in the following Tab. 6.7.

**Tab. 6.7:** Biographical details of PETE and MARC

| **Biographical details of the interviewee (Pete)** | |
|---|---|
| Synonym | PETE |
| Year of birth | 1972 |
| Age | 36 |
| Nationality | German |
| Professional experience | 9 years |
| University / Graduation | Geo informatics, 1999 (discontinued) |
| **Current employment** | |
| Role | Consultant for software solutions |
| Time of employment | 2000 until today |
| Job description | Effort estimation, project lead, support and maintenance |
| Completed projects | 10 – 15 |

| **Biographical details of the interviewee (Marc)** | |
|---|---|
| Synonym | MARC |
| Year of birth | 1971 |
| Age | 37 |
| Nationality | German |
| Professional experience | 11 years |
| University / Graduation | Business administration, 1996 |
| **Current employment** | |
| Role | Head of software solutions |
| Time of employment | 2005 until today |
| Job description | Head of software solutions, effort estimation, project lead, team lead, personnel responsibility, key account manager, IT process manager |
| Completed projects | 10 |

The company PETE and MARC work for is outlined in the following Tab. 6.8.

**Tab. 6.8:** Employer of PETE and MARC

| **Company Details** | |
|---|---|
| Synonym | *Objects Inc.* |
| City, Country | (undisclosed), Germany |
| Turnover per year | ~ 20 million € |
| Employees | ~ 170 (25 software developers) |
| Market | Energy trade and supply |
| Core competence | IT Service, Process outsourcing for energy suppliers |
| Typical customers | Energy suppliers and public transportation companies |
| Project members | 1-30 |
| Project volumes | ~ 10,000 € (small); ~ 5 million € (large) |

The process of effort estimation was explained by MARC:

> *Effort estimation is based on experience. Additionally, we discuss each project with colleagues who are experts for certain topics. Generally, effort is estimated in man-days. We focus on functional requirements in order to produce estimates. We try to consider worst cases, but we rather concentrate on the most likely case. Besides, we adjust the estimates by surcharges for project manage-*

> *ment, for example. Finally, the estimates are summed up and determine the project costs that will be then given to the customer.*

In view of that, effort estimation is an expertise-based technique, which satisfies assumption A1. Furthermore, estimates are produced early and determine overall project costs of which the project owner gets informed. Thus, assumption A2 is also supported. With respect to assumption A3, PETE explained:

> *The project status is tracked by the central project office, which monitors all projects at Objects Inc. The project leaders continuously report on the project status, including the current budget overview as well as qualitative information. The project office reacts to overruns. Typically, the project plan is reworked and changes are discussed with the customer. Work packages are removed only if forced by circumstances, for example, if the integration of a third party system is much harder than expected. However, in this case, work packages are rather replaced by workarounds.*

Accordingly, the project progress is monitored and project lead takes action if necessary. This supports assumption A4. On the question whether or not he is affected by the PARKINSON effect, PETE only answered:

> *"No, I am always done on time."*

Since he evaded further questions, his answer allows three conclusions. First, PETE told the truth and his work performance is independent of given estimates. Second, PETE could not imagine to be affected by the PARKINSON effect. Third, he did not want to admit this in presence of his boss. However, MARC chimed in:

> *"We often exceed deadlines."*

With regard to conforming to deadlines, they were in complete agreement. Both stated that one could not work faster in order to meet deadlines. However, they explained that developers either work overtime or they stop others from disturbing them. Accordingly, developers at least conform to estimates to meet deadlines. PETE'S response allows concluding that he is unconsciously affected by the PARKINSON effect. Always finishing work packages on time would indicate the constructive form of PARKINSON (see Section 4.3.6), so that PETE uses the estimates to organize and interpret work packages appropriately. MARC'S comment suggests procrastination and the PARKINSON effect as well. Therefore, their responses allow concluding that developers are not independent of given estimates. Thus, assumption A4 is satisfied.

Furthermore, PETE and MARC agreed that there are perceptible individual differences between developers. While PETE estimated a productivity factor of 1:2 between developers, MARC gave a higher factor of 1:4, which satisfies assumption A5.

Again, all assumptions are affirmed by the interview results. Projects at *Objects Inc.* seem to be affected by estimation fulfillment. Regarding the deviation of actual and estimated effort, MARC explained:

> *I think projects exceed estimated effort by 10 to 20%. The maximal common deviation is +40%. In the past, there were, of course, projects that were finished with +100%. These are exceptions. Conversely, there were projects that were clearly overestimated. These projects could be completed with -30% of the estimated effort.*
>
> *Generally, schedule overruns are worse. Overruns of 40 to 50% are no surprise. Projects that should be completed in one year require 18 months, for example.*

With reference to effort underruns, he continued:

> *If projects are dominated by unfamiliar or new requirements, effort will frequently be exceeded. We are not happy about that. If projects allow routine and analogies, effort estimates are better. However, except for projects of which we know that they were clearly overestimated, actual effort never falls below estimated effort.*

Accordingly, projects at *Objects Inc.* generally exceed the estimates by a moderate percentage. As explained by MARC, effort underruns are an exception and caused by obvious overestimation. This is in line with the propositions P1 and P2. For that reason, the findings of expert interview #4 also corroborate the EFT.

Finally, with regard to individual differences of developers and their consideration during effort estimation, MARC explained:

> *We use a human resource development model. This is based on a skill matrix with a four-point rating scale. An entry describes one particular skill, for example, a programming language. The list of entries is standardized.*
>
> *However, we do not explicitly draw on this model during effort estimation. We rather ask experienced developers to adjust our estimates and to consider the skill sets of potential developers. Moreover, work packages are usually assigned to specialists. Besides, we have special budgets for on-the-job trainings, which compensate budget overruns caused by novice developers.*

### 6.2.5 Expert Interview #5

The interviewee of the fifth expert interview was MARCUS who works for *Newtec Inc*. This German company focuses on individual software solutions as well as IT consulting. *Newtec Inc.* is not limited to special business domains. Typical customers are German medium-sized businesses as well as DAX-listed companies.[754] The interview was conducted on March, 20th 2009 and took two hours. Biographical details of MARCUS are given in the following Tab. 6.9.

**Tab. 6.9:** Biographical details of MARCUS

| **Biographical details of the interviewee** | |
|---|---|
| Synonym | MARCUS |
| Year of birth | 1977 |
| Age | 31 |
| Nationality | German |
| Professional experience | 5 years |
| University / Graduation | Münster, Information Systems (MScIS), 2003 |
| **Current employment** | |
| Role | Software developer |
| Time of employment | 1/2007 until today |
| Job description | Software development, Maintenance, Effort Estimation, Specifications, Testing |
| Completed projects | 2 |

Since 2007, MARCUS has been employed by *Newtec Inc*. Besides software development and maintenance, he is also concerned with effort estimation. His employer *Newtec Inc.* is briefly outlined in Tab. 6.10.

**Tab. 6.10:** Employer of MARCUS

| **Company details** | |
|---|---|
| Synonym | *Newtec Inc.* |
| City, Country | (undisclosed), Germany |
| Turnover per year | ~ 10 million € |
| Employees | ~130 (including ~100 software developers) |
| Market | Project software, resource planning software, service management |
| Core competence | Web-based solutions, individual business solutions, Java |
| Typical customers | Midsize and large, DAX-listed companies |
| Project members | 1 – 15 |
| Project volumes | Small projects: 10 man-days, midsize projects: ~6 man-month, large projects: ~3 man-years |

The effort estimation approach used at *Newtec Inc*. was described as follows:

[754] *DAX (German abbr.): German Stock Index ("Deutscher Aktienindex").*

> *First, we collect all requirements. Usually, we have the functional specification document at hand. Thus, the software system can be divided into modules. Likewise, the development process can be divided into small work packages. For each work package, we estimate effort by analogy and gut feeling. Next, we summarize all estimates in order to get the total number of man-days. For new development projects we add an extra risk surcharge. Next, we distribute effort to design, implementation, testing, quality assurance etc. After the estimates are produced and we have decided on the percentages for the different development phases, we turn to the master project plan.*

In view of that, effort estimation at *Newtec Inc.* makes use of analogy, intuition, as well as gut feeling. Hence, assumption A1 is supported. Moreover, since MARCUS describes that estimates are produced after an initial requirements analysis and before the conception of the project plan, assumption A2 is also satisfied. With respect to assumption A3, he explained:

> *We have a time tracking system based on our internal ERP system. Working time is accounted on the work package level. This data is used by project lead to monitor subprojects and track progress. The status of a subproject is reported to the chief project lead. If a subproject gets behind schedule, they will decide whether to remove work packages or to move certain packages into later releases. However, they usually inform the customer about their decisions.*

Accordingly, project lead conforms to the given estimates. Therefore, assumption A3 is supported. Concerning the question if developers also conform to estimates, MARCUS continued:

> *I think, the PARKINSON effect applies since developers always find ways to optimize or beautify a solution. I have to point out that most of our projects are fixed-price projects. However, we also do "body leasing." In this case, we officially work for the customer and the entire working time is directly accounted. Therefore, developers will always fill available time since the customer has to pay for it.*
>
> *In contrast, if developers have to meet calendar deadlines, they cannot just work faster. Maybe you can increase your productivity by 20% for a short time by working focused and concentrated. I think, however, that meeting calendar deadlines generally leads to overtime.*

> *Nevertheless, while I am completing a work package, I definitely consider the given effort estimate. The estimate is as important as the functional specification. You generally work against time. So, it is the combination of specification and estimate, which determines the solution.*

This explanation reflects that developers also conform to estimates. Therefore, assumption A4 is satisfied. Besides, the previous explanation contains an interesting differentiation of project types. All interviewees implicitly referred to *fixed-price projects*. MARCUS, however, pointed out that for some projects at *Newtec Inc*. prices are not fixed upfront and, for that reason, all spent effort will be billed. For this project type, actual effort will never be less than estimated, and, thus, one will always find estimation fulfillment. However, the interests and incentives of all involved stakeholders are different compared to fixed-price projects.

With regard to assumption A5, the aspect of individual differences across developers was described as follows:

> *There are noteworthy differences between developers. However, lots of them represent the average. Besides, there are some very good developers as well as some botchers who get nothing done. [...] I think low qualified developers work at 80% of the average while good developers work at 130%. [...]*
>
> *Over time, my domain knowledge has increased most. Technically, I would say I stayed on the same level, for example, in terms of programming skills. I think my overall productivity raised by 300% when I consider the increase of domain knowledge.*

Accordingly, assumption A5 is supported. In consequence, all five assumptions are satisfied. The projects MARCUS was involved in are supposed to be affected by estimation fulfillment. Regarding the deviation of estimated effort, he explained:

> *Small projects are completed around ±10%. Some of them are above, some below the estimate. For example, a change request is estimated to take one day, but it was completed by one developer in 5 hours. Conversely, large projects are arranged to fit the estimates. However, these projects still exceed the estimates by approximately 10% or more.*

Thus, if small projects are excluded (see interview #2), the propositions P1 and P2 are both supported. This interview shows once more that small projects are special and cannot be included into the theory. Nevertheless, as all assumptions and propositions are satisfied, the findings from this expert interview corroborate the EFT.

Concerning proposition P3, MARCUS noted:

*We use excel spreadsheets to document staff skills. However, the spreadsheets are based on self-assessment. The spreadsheet asks about technological aspects, for example, J2EE, SQL, Maven etc. Each developer answers whether he or she 'has no idea,' knows 'only theory,' or is an 'expert.'*

### 6.2.6 Expert Interview #6

The sixth expert interview was conducted on April, 1st 2009 with HALE who is head of development at *BankIT*, which is already known from expert interview #3 (see Section 6.2.3). HALE has been employed by *BankIT* since 1996. He has been concerned with *all* projects during his employment. He estimated that BankIT runs 20 to 30 projects per year. The interview with HALE took 2.5 hours. Biographical details of the interviewee are given in the following Tab. 6.11.

**Tab. 6.11:** Biographical details of HALE

| **Biographical details of the interviewee** | |
|---|---|
| Synonym | HALE |
| Year of birth | 1969 |
| Age | 39 |
| Nationality | German |
| Professional experience | 15 years |
| University / Graduation | Fachhochschule Steinfurt, Mechanical Engineering, 1994 |
| **Current employment** | |
| Role | Head of development |
| Time of employment | 1996 until today |
| Completed projects | All; ~ 20-30 per year |

As head of development, HALE takes an important part in the effort estimation process at BankIT. Besides, he is concerned with a recent redesign of that process. HALE explained:

*We start with the requirements document, which, of course, contains all known requirements and features. We use this document to start working on an effort spreadsheet. Therefore, we break down the project into work packages, which have a maximal net duration of five days. Three or four senior staff members are then assigned to effort estimation. Estimates are generally based on experience and intuition. Finally, we consolidate the results in order to get averages.*

*We recently designed a new estimation approach and we are still working on it. We call it "estimation poker." When the list of work*

> *packages is finished, we arrange a meeting with experienced, senior managers as well as the project owner. In this meeting, we run through the list of work packages. For each work package each participant chooses a playing card. A playing card has a number, which represents man-days. When all participants have cards, they must lay them down on the table. Then we start a group discussion in order to reach consensus on the estimate. Since all cards stay on the table, no one can change his opinion without discussing and justifying his or her decision.*

Accordingly, it is confirmed again that effort estimation at BankIT is an expertise-based approach. The recently designed "estimation poker" has similarities with Wideband Delphi. Assumptions A1 and A2 are satisfied. HALE continued:

> *We record effort estimates, as they are part of the project plan. Moreover, we have a time tracking system that is used for projects. Based on this data, we track project progress daily in order to recognize trends. [...]*
>
> *We recently started to use ranges instead of single values. Thus, the result of the estimation process is a range of effort. During the project, we aim at the lower limit of that range. If we see that we can reach the minimum or that we are too fast, we include further requirements.*

The previous explanation of HALE describes how project lead reacts on the project progress, which supports assumption A3. With respect to the PARKINSON effect and deadlines, HALE answered:

> *Yes, developers fill available time with gold-plating. Regardless which time limit or budget is set, it will be expended.*
>
> *In contrast, when deadlines have to be met, I think developers cannot simply increase workload. They rather have to do overtime or weekend working.*

Accordingly, assumption A4 is supported. On individual differences between developers, HALE commented:

> *There are differences, but they are not immense. I think the overall productivity difference between the most experienced and a novice developer is 50%.*

Although this productivity ratio must be taken as low, it still allows a notable variation of effort. Moreover, HALE stated that developers do not necessarily differ in general productivity, but they differ in fields of expertise, so that the body of knowledge is spread over the entire developer team. Thus, assumption A5 is satisfied. With respect to the deviation of actual and estimated effort, HALE explained:

> *As said before, we use risk buffers of 15 to 20% for each project. However, buffers are always used up. Next, there are two types of projects. The first type has clear, comprehensible specifications. For this type, the deviation of actual and estimated effort is around 30 to 40%. The second type is characterized by unclear, vague, or ambiguous specifications. Here, effort overruns of up to 100% are probable. In opposition, effort underruns are a rare exception.*
>
> *Generally, there are always deviations in terms of budget, scope, and schedule. According to my personal experience, I have to say that "approved budgets are always gone."*

In view of that, the propositions P1 and P2 are supported. Since all assumptions and propositions hold true, the findings from this expert interview corroborate the EFT.

With regard to proposition P3, Hale explained:

> *We do not use profiles. "You have to know who's good at what." This knowledge is mainly based on appraisal interviews with developers in which we agree on objectives.*

### 6.2.7 Expert Interview #7

The seventh interview was conducted on April, 16th 2009 with KAY and CHRIS who work for *Performix Inc.,* a company that offers *Enterprise Content Management* (ECM) and *Corporate Performance Management* solutions. The company focuses on software development and consulting in equal shares. Typical customers are German medium-sized businesses as well as larger DAX-listed companies. *Performix Inc.* does not focus on special business domains. The group interview took two hours. Biographical information of the two interviewees is given in the following Tab. 6.12. A brief summary of *Performix Inc.* follows in Tab. 6.13.

**Tab. 6.12:** Biographical details of KAY and CHRIS

| **Biographical details of the interviewee (Kay)** | |
|---|---|
| Synonym | KAY |
| Year of birth | 1959 |
| Age | 49 |
| Nationality | German |
| Professional experience | 15 years |
| University / Graduation | Computer Linguistics, 2004 |
| **Current employment** | |
| Role | Head of Enterprise Content Management |
| Time of employment | 2008 until today |
| Job description | Project lead, effort estimation, personnel responsibility |
| Completed projects | 4 projects at *Performix Inc.* (~ 75 projects at former employers) |

| **Biographical details of the interviewee (Chris)** | |
|---|---|
| Synonym | CHRIS |
| Year of birth | 1980 |
| Age | 28 |
| Nationality | German |
| Professional experience | 2 years |
| University / Graduation | Information Systems, 2006 |
| **Current employment** | |
| Role | Enterprise Content Management Developer |
| Time of employment | 2007 until today |
| Job description | Software development |
| Completed projects | 6 |

**Tab. 6.13:** Employer of KAY and CHRIS

| **Company Details** | |
|---|---|
| Synonym | *Performix Inc.* |
| City, Country | (undisclosed), Germany |
| Turnover per year | ~ 10 million € |
| Employees | ~ 100 (including 20 software developers) |
| Market | Software development, IT consulting, solution provider |
| Core competence | Enterprise Content Management, Corporate Performance Management, Database Consulting |
| Typical customers | German medium-sized businesses, DAX-listed companies; no special business domain |
| Project members | 3-4 |
| Project sizes | Three classes: 1) 70-80 man-days, 2) 100-300 man-days, 3) up to 1.000 man-days |

In the following, the statements given by KAY and CHRIS exclusively refer to ECM projects since they do not collaborate in projects that address Corporate Performance Management solutions. The process of effort estimation was explained by KAY:

> *We estimate effort on the basis of professional expertise. Therefore, we search our staff for contact persons who have appropriate competences. Effort estimation is generally a mixture of experience and gut feeling. We try to find analogies or reference values. I think that "effort estimation is located somewhere in a gray area of cognition and perception."*
>
> *Generally, we begin with an analysis of requirements and regularly use UML modeling. However, we produce estimates before turning to fine-grained specifications. In consequence, estimated effort and budget become fixed values for fixed price projects.*

Accordingly, effort estimation is based on an expertise-based approach, which supports assumption A1. Since estimates are produced before detailed specifications are created, assumption A2 is also supported. With regard to the assumptions A3 and A4, CHRIS explained:

> *You can work faster if you are put under pressure. However, this option is limited in time. Typically, if project lead expects a follow-up project, they will tell us: "Make it work first – you can make it nice later." Thus, we move parts of a work package into the next project. Moreover, project lead might change the project plan so that we have to meet a calendar deadline. Usually, this is done by overtime. Nevertheless, project lead takes care of the project progress and intervenes if anything goes wrong.*
>
> *During development, a Scrum sprint is processed as follows: Project lead maintains a backlog with prioritized work packages. Depending on the team size and the available time, project lead determines how much effort can be assigned to the team. Next, we – the team – estimate effort for each work package. These estimates become target values. During the sprint, we process work packages according to their priority.*
>
> *"Since we don't like to work for nothing," we want the actual total effort to be in line with the initially estimated effort, on which the project price is based.*

In view of that, project lead monitors projects and takes actions if necessary. During a Scrum sprint, developers produce their own estimates, which they use as target values. Therefore, they conform to these estimates. The development process design most likely causes delays at later project stages since work packages are dynamically assigned to teams and sprints. Due to the dynamic and flexibility between project lead and developer teams, both sides are supposed to conform to estimates, and, hence, assumptions A3 and A4 are satisfied. With respect to individual differences between developers, KAY stated:

> *There are differences and they are elementary. "I think there is something like a developer gene. You simply have it or not."*

CHRIS explicitly agreed on KAY'S statement. Moreover, both KAY and CHRIS estimated that the general productivity ratio between their colleagues is around 1:4. Accordingly, assumption A5 is supported.

As all five assumptions are satisfied, the projects CHRIS and KAY participated in are supposed to be affected by estimation fulfillment. The deviation of actual and estimated effort was explained as follows:

> *Actual and estimated efforts strongly deviate. I think the larger projects are, the stronger they deviate. However, things got better during the last decade. Today, effort overruns of 100% are extreme. Ten years ago, I saw projects with overruns of 300%. Nevertheless, good projects are completed with effort overruns of 10 or 15%. I think typically we finish projects with overruns of 20%.*
>
> *And we never finish large projects with effort underruns. In contrast, small projects can be completed while falling below the estimates. However, small projects are different because they only have one Scrum sprint. Therefore, it can happen that developers have a good day and finish ahead of schedule. Larger projects with more than one sprint have never been completed with effort underruns.*

This statement contains an interesting counter-finding that helps stating the theory more precisely. On the one hand, proposition P1 is supported since projects typically end with effort overruns of approximately 20%. On the other hand, effort underruns are also given, which again contradicts proposition P2. However, effort underruns have only been achieved for *small* projects. In this case, projects were characterized as small if they are completed in one Scrum sprint. For large projects, covering more than one sprint, proposition P2 is supported. This confirms that the project size must be incorporated into the theory as a limitation.

## 6.3 Discussion of Findings

Interestingly, all effort estimation approaches described by the interviewees belong to the class of expertise-based techniques. None of them used parametric models, Function Points Analysis, or variants of the FPA. This might be an indication that

HEEMSTRA'S survey results (see Section 2.4) still apply for software development companies today.[755] Some of the interviewees explained that they heard of Function Points Analysis, for example, during their undergraduate studies. One interviewee named the 3-Point Estimation technique. However, none of the interviewees knew COCOMO or Delphi.

Nevertheless, all interviewees make use of work-breakdown techniques. Both the software system as well as the development process is broken down to smaller elements, for example, work packages. This is always done at an early project stage and usually based on the initial requirements document. Afterwards, effort is estimated per work package. For this process, all companies rely on experience, analogy, gut feeling, and/or intuition.

The total estimated effort is given as a single value. Only one interviewee explained that his company recently changed the effort estimation process in order to use ranges. Using ranges instead of single values is highly recommended, for example, by MCCONNELL, since ranges better reflect project risks and make the process of including or excluding requirements more transparent.[756]

Generally, the estimation processes described by the interviewees do not rely on single persons. All approaches are based on either group decisions, the consolidation of multiple estimations, review processes, and/or the consultation with business experts. However, the interviewees neither explained why they designed their approaches as they are nor why they decided for or against particular process steps. Similarly, none of the interviewees referred to theoretical foundations or techniques that are discussed in literature. One exception is given by interview #6, because the interviewee described a newly introduced estimation technique, which they call *"estimation poker."* Although they use a slightly different name, this technique is known as *"planning poker,"* especially in the context of agile software development.[757]

The estimation approaches used by the interviewees appeared to have evolved in the past until they have become established. This might explain why some of the interviewees pointed at conflicts between their established estimation technique and their recent decision for iterative development.

Almost all projects run by the interviewees' companies had fixed prices. One exception was given by interview #5, as the corresponding company offers labor leasing.

755 HEEMSTRA (1992).
756 MCCONNELL (2006).
757 E.g., COHN (2005).

In this case, the customer has to pay for all invested effort and, as a consequence, developers always fill available time. Therefore, this special case was excluded from the interview.

Since the initial effort estimation determines the project costs as well as the final price, effort estimates become target values, which must be met in order to make profit. Correspondingly, none of the interviewees described or asked for an independency between effort estimates and actual project effort. Estimates are rather used and communicated as if they were precise calculations. This is in line with the fact that five of six companies favor single-valued estimates over ranges.[758]

Project lead generally monitors projects and tracks progress in order to intervene if necessary. For the interviews #2 and #3, the actions of project lead were not discussed in detail. However, according to the explanations of these two interviewees, project lead regularly monitors projects and discusses the project status with developers. This suggests that project lead has the option of taking necessary actions. Moreover, the interviewees #3 and #6 work for the same company. As mentioned above, the corresponding company, *BankIT*, has recently started to use ranges. Interviewee #6 explained that they dynamically include or exclude requirements in order to meet the estimated range. Similarly, interviewee #2 explained that developers go for an 80% solution in order to meet deadlines. This decision, however, is supposed to be made by project lead. Therefore, the assumption A3 is regarded as supported by the interviews #2 and #3 as well.

In consequence, all interviews confirmed that project lead tries to conform to approved estimates during a project. Typical actions that were noted by the interviewees are putting pressure on developers, initiating overtime, weekend working and nightshifts, including and excluding requirements, reprioritizing work packages, changing the project plan, as well as moving work packages into follow-up projects.

One interviewee referred to the problem of mixed effort, budget, and schedule. Therefore, a precise tracking of project effort while the project is running is difficult. Precise totals are only available after the project is completed.

Nevertheless, all interviewees agreed and confirmed that developers also conform to estimates. Procrastination, the Parkinson effect, as well as gold-plating were considered as plausible drivers. Besides, the interviews showed that developers also use estimates as target values since they generally work against time. Since developers are usually not rewarded for being ahead of schedule, they try to use the available

[758] Two of the seven interviews were conducted at the same company (*BankIT*).

time to optimize, beautify, and test a solution. In consequence, most work packages are delivered on time. Due to technical problems, high complexity, underestimation, or high difficulty, some work packages exceed the estimates. In consequence, projects also exceed the total estimated effort.

Concerning the individual performance of developers, all interviewees agreed that there are significant differences across developers. The responses of the interviewees open two dimensions in this context. First, there are differences in terms of overall productivity. Second, developers have different domains of knowledge, experience, and expertise. While some interviewees reported productivity ratios of up to 1:10, some interviewees noted that overall productivity might not necessarily be different, but some developers are limited to a small area of expertise. A common response was that effort estimation would be much easier if developers were similar in terms of capability and experience. However, good developers are rare and always sought-after.

Interestingly, concerning the question if individual differences of developers are explicitly taken into account during effort estimation, KAY (interview #7) stated that he tries to break this habit. He explained that teams are usually heterogeneous and that the consideration of individual differences lowers the overall effort for a team, since good developers need less time for solutions. However, he noted that the consideration of differences will become a risk if good developers drop out, for example, because of other important projects, illness, or resignation. He prefers to promote learning from each other and to give good developers time for helping their colleagues.

Each of the seven interviews supports all five assumptions. Therefore, the corresponding companies are supposed to be affected by estimation fulfillment. Proposition P1 (effort overruns) was supported by all interviews without any exception. The reported mean effort overruns range from 10 to 40% (see Tab. 6.14).

**Tab. 6.14:** Summary of interview findings

| Interview | Company | A1 | A2 | A3 | A4 | A5 | P1 | Overrun | P2 | Counter-finding |
|---|---|---|---|---|---|---|---|---|---|---|
| #1 | *EnSwiss* | + | + | + | + | + | + | 20% | + | / |
| #2 | *SolutionFactory* | + | + | +/d | + | + | + | 20-30% | + | Small projects |
| #3 | *BankIT* | + | + | +/d | + | + | + | 20% | + | / |
| #4 | *Objects Inc.* | + | + | + | + | + | + | 10-20% | + | / |
| #5 | *Newtec Inc.* | + | + | + | + | + | + | 10% | + | Small projects |
| #6 | *BankIT* | + | + | + | + | + | + | 30-40% | + | / |
| #7 | *Performix Inc.* | + | + | + | + | + | + | 20% | + | Small projects |

*+ = supported, d = debatable, / = no counter-findings*

Conversely, with regard to proposition P2 (effort underruns), three interviews (#2, #5, and #7) had contradicting results. However, the corresponding interviewees described identical situations in which effort underruns have been observed in the past. They all agreed that effort underruns only happened for *small* projects, which are clearly different from midsize or large projects.

According to MARCUS (interview #5), small projects have a limited effort of approximately ten man-days. Similarly, CHRIS and KAY (interview #7) characterized projects as small if they are processed within a single Scrum sprint. For small projects, positive and negative deviations of actual and estimated effort do not compensate, since these projects only have a few work packages. Therefore, some small projects are completed with less effort than estimated. MARCUS gave a simple example: *"A change request is estimated to take one day, but it was completed by one developer in 5 hours."* Accordingly, the project size must be used to define a limitation that excludes small projects from the theory.

Finally, all seven interviews corroborated the EFT. For that reason, no further expert interviews were conducted. Previously, two interview series with ten interviews per series were planned. Of the first series, three interviews were not arranged after a preliminary talk, as the interviewees had only little experience with effort estimation. A second series was regarded as necessary, because it was expected that less interviews support the assumptions. However, since all interviews of the first series supported the assumptions, the contribution of a second interview series was strongly doubted, and, therefore, not conducted. It has to be noted that the first expert interview series refers to project experiences based on at least 330 software development projects.

## 6.4 Expert Review

The expert review was conducted in order to allow an evaluation of the EFT, i.e., the underlying causal model, its assumptions, propositions, as well as the presented estimation fingerprints from the practitioner's perspective.[759] The goal of the expert review is to determine whether or not practitioners can comprehend and agree upon the theory construction. Besides, it could be discussed whether or not the findings from the expert interviews were correctly understood and led to acceptable conclu-

[759] The conduction of an expert review is motivated by the recommendations of HEVNER *et al.* (2004) concerning the evaluation of newly designed artifacts in the context of design science research. A related evaluation approach is the use of *focus groups* as discussed by GIBSON, ARNOTT (2007).

sions. The expert review was conducted with the first interviewee, BOB, in April 2009 after the EFT was completely constructed and written down. The expert review took 2.5 hours.

### 6.4.1 Review of Causal Relationships

At first, the supposed causal relationships were discussed as given in Section 5.1.5. Each point was separately addressed. The points 1 to 4 were accepted immediately. With reference to point 5 (*"Estimates (unconsciously) become target values"*), BOB explained:

> *In the beginning, there is always just a basic project idea. We have to get budget first. Thus, we must produce rough estimates based on a coarse-grained breakdown of the project idea. This is done using experience and analogy. We use day rates and – if known – additional hardware or license costs in order to estimate the required budget, let's say 10 million CHF.*
>
> *If the budget is approved, we will initiate a kick-off workshop, which normally lasts two days. In this workshop, we discuss the project with business experts in order to develop more precise business requirements. The result is the project scope represented by a project initiation concept or business concept respectively.*
>
> *Now, we hand over the business concept to the business experts who must proceed with business requirements engineering in order to gain detailed business requirements. This point is exactly where estimates have already become target values. Business experts are strongly influenced by or even forced to conform to the given project scope and the approved budget. At this point, estimates have become guidelines for the project.*

Accordingly, estimates can already become target values on the planning and management level of a project before any developer gets in touch with the project and even before any work package gets specified in detail. The initial project estimation used for budget approval influences the project scoping, which in turn influences any detailed specifications.

Point 6 (*"Project lead tracks progress"*) was also confirmed. BOB added:

> *Of course, we monitor how a project evolves in order to intervene. "A project plan is a living construct." Typically, we have to communicate pressure of time and remind developers of the importance of milestones. If the project gets behind schedule, we often have to rework the project plan. We must resolve resource conflicts, repri-*

> *oritize work packages, or even discard minor requirements in order to go for an 80 or 90% solution. Above and beyond, calendar deadlines and estimated effort must be seen differently. If the project is about to miss a calendar deadline, we order overtime, weekend working, or nightshifts through the disciplinary superiors. This, of course, causes effort overruns for the corresponding work packages.*

With reference to point 7 (*"Estimates also become target values for developers"*), BOB pointed out:

> *We cannot disregard the fact that a few developers want to perform and evolve. They complete work packages ahead of schedule. This type of developer, however, is rare. Other developers use available time for self-fulfillment. They implement functionality that is not required, but that does not bother anyone either. Nevertheless, the idea of procrastination, gold-plating, and the interpretation of estimates as part of the specification are plausible and can definitely be found in practice. I think that developers generally spend more time than estimated. Therefore, they mainly conform to the estimates because project lead responses to the chronic tendency of exceeding estimates.*

Accordingly, point 8 (*"Projects generally tend to exceed the estimated overall effort"*) was also confirmed. BOB commented on point 9:

> *Effort underruns are a rare exception. Effort overruns of 20%-25% are typical.*

Finally, with regard to point 10 (*"The consideration of individual difference reduces the deviation of estimates and actual efforts"*), BOB stated:

> *I think this is true. We have to give shorter estimates to good developers in order to minimize overall effort. If we did not know who can perform well, we would not be able to do so. On the contrary, we have to take into account that some developers are not very experienced. Otherwise, our project plan would include predefined effort overruns. I think that ignoring individual differences of developers causes higher deviations of estimates as well as overall actual effort. If we did not consider the skill sets of our developers, we might face average effort overruns of 40% or more, which would not be acceptable.*

### 6.4.2 Review of Assumptions, Propositions, and Central Theory Statement

Subsequent to the review of the causal model, the assumptions as given in Section 5.2.2 were discussed. With respect to readability, the assumptions will be repeated in footnotes. Assumption A1 was accepted immediately.[760] With regard to assumption A2[761], BOB explained:

> *"Getting budget for a project is not based on wishful thinking." We have to convince the higher management of the project. Therefore, we need a good plan that contains early, rough estimates. Otherwise, we get no budget. In consequence, this assumption will always be satisfied for our company.*

Concerning assumption A3[762], BOB added:

> *A project leader is forced to fulfill the project plan. That's his or her job. Especially milestones are important. He or she should have an intrinsic motivation that the project plan and the estimates get fulfilled.*

Correspondingly, BOB commented on assumption A4[763]:

> *Most developers do not volunteer to conform to the estimates. Some developers are simply bellyachers. I think half of them always say that they need more time, or that the specification cannot be implemented in the given timebox. They rather commit to the estimates since they receive orders and they do not want to get into trouble. However, pressure is mandatory. Otherwise, the project must be cancelled right after project kick-off.*

Finally, assumption A5 was confirmed as plausible.[764]

---

[760] Assumption A1: Practitioners favor expertise-based techniques, i.e. Work Breakdown Structures, Estimation by Analogy (or intuition), and/or (Wideband) Delphi.

[761] Assumption A2: Expertise-based techniques allow early estimates based on premature requirement specifications. Accordingly, if practitioners estimate effort by expertise-based techniques, estimates are produced and communicated at early project stages.

[762] Assumption A3: Project lead tracks progress and takes necessary actions in order to prevent the project from getting behind schedule. Thus, project lead conforms to the given estimates.

[763] Assumption A4: Developers tend to procrastinate, fill available time with gold-plating activities, and/or perceive effort estimates as an essential part of work descriptions. Estimates, therefore, significantly influence how developers interpret and process work packages. As a result, developers conform to effort estimates.

[764] Assumption A5: The productivity of developers significantly varies due to individual differences, e.g., different levels of capability, experience, and learning plateaus.

After reviewing the propositions, BOB assessed all three propositions as plausible.[765] Moreover, he added that they hold true according to his project experience. With respect to proposition P3, he explained:

> *For lots of developers, the determination of their skill sets is rather a "best guess." The skill sets work as reference values. They can also be wrong or biased, for example, because one developer was performing exceptionally well when his or her skills were discussed. Later, however, the developer showed worse performances in subsequent projects. From our point of view, it is almost infeasible to gain more precise information about the skills of our developers.*
>
> *Nevertheless, I think our development environment is rather constant. We stay in the same domain, we have similar project owners, and we do not change technology frequently. Our staff is also constant. Therefore, I believe the central question for effort estimation is "Who does the work?" Our approach helps answering this question. If we could exactly predict the skills of our developers, we would finish projects on time. There would be no deviation except for projects that are subject to a "force majeure." Generally, I agree that the remaining, unconsidered individual differences must have the greatest impact on effort overruns.*

With regard to the central statement of the EFT, BOB noted:

> *The statement is comprehensible and plausible. To put it in other words, one can also summarize the theory by saying "I know that my boss won't bite my head off if the project estimate is exceeded by 20% or 30%."*

Finally, the estimation fingerprints were evaluated as reasonable. BOB noted that both the fingerprints referring to underestimation and continuous overestimation are unlikely to be found in practice. Concerning the estimation fulfillment fingerprint, BOB argued that the fingerprint is most probably right-skewed.

[765] Proposition P1: Except for rare outliers, actual project efforts will not fall below estimated efforts if the project is an EF-type project.
Proposition P2: Actual project efforts will exceed estimated efforts by a moderate percentage, i.e. 20%-30%, on average, if the project is an EF-type project.
Proposition P3: If individual differences between developers are systematically and precisely documented and taken appropriately into account during effort estimation, the deviation of estimated and actual effort will reduce for EF-type projects.

# 7 Conclusion

This chapter concludes the thesis. First, the research process including the central research questions is recapitulated. Next, the major contributions are summarized. Section 7.3 details the theory's implications for research and practice. Afterwards, Section 7.4 explains the limitations of this study. Finally, an outlook to future research is given in Section 7.5.

## 7.1 Reprise

This thesis is motivated by the relevance of understanding and conducting effort estimation in software development projects. Whilst other research focuses on the development, improvement, and alignment of estimation approaches, this thesis addresses the behavioral impact of effort estimates on actual project effort. The question whether estimated and actual efforts are dependent or independent is of utmost importance in order to understand the behavior of developers, the fundamental mechanisms in software development projects, as well as to understand and contribute to discussions of estimation accuracy. Accordingly, two research questions were addressed:

1) Is it reasonable to assume a general independency of estimated and actual software development effort?

2) How do developers and project lead influence the relationship between estimated and actual software development effort?

In order to answer these questions, a multi-method-based study was designed. The study design starts with an exploratory study using empirical data, statistical data analysis, as well as simple random subsamples. The results of this exploratory study, combined with previous research and the subjective previous knowledge of the researcher, led to the conclusion that it is *not* reasonable to assume a general independency of estimated and actual software development effort. This insight formed the basis for the development of a new theory that focuses on both explanation and prediction (EP theory). The new theory explains how and why estimates influence actual project efforts. Additionally, the theory explains under which circumstances companies are affected by estimation fulfillment, and what the economic consequences are. The theory was tested by semi-structured expert interviews conducted with software professionals in order to corroborate the theorized propositions. The responses of the experts confirmed all propositions. Additionally, an expert review was conducted in order to validate the acceptability and comprehensibility of both the theory and its development from the practitioners' perspective.

## 7.2 Contributions

This thesis contributes to the body of knowledge of software engineering by discussing and gathering knowledge about one particular, but essential aspect of development effort estimation. A new theory was developed which is capable of explaining why and predicting how particular software development projects evolve. The Estimation Fulfillment Theory states that – under defined circumstances – projects will generally end with moderate effort overruns, which is not the result of insufficient estimation accuracy. Except for rare outliers, projects will not go below the estimated effort. Small projects are a general exception, and, in consequence, they are excluded from the EFT. According to the expert interviews, a project can be classified as small if it covers only one scrum sprint or if estimated effort is less than a certain number of man-days. This classification is supposed to vary between companies. Nevertheless, estimation fulfillment is assumed to be natural in software development projects. Even more, it can be understood as an important enabler for successful project completion.

The study makes several contributions. As the central contribution, it offers a comprehensible, empirically corroborated EP theory that describes dependence between estimated effort and actual development effort. In addition, the theory provides a causal model, as well as a set of assumptions and propositions. For that reason, it explains *"why"* companies can generally be affected by estimation fulfillment, *"when"* a particular company or project is affected, and which economic consequences *"will"* manifest. The theory helps understanding the underlying mechanisms of software development projects and offers a new perception of and participation in discussions of effort estimation techniques and estimation accuracy. Perhaps most importantly, the theory *gives a name* to an extrinsically simple, but intrinsically complex phenomenon.

Besides this central contribution, additional contributions are summarized in Tab. 7.1. This overview assigns contributions to different phases of this study as well as different types of contribution, which originate either from the applied methodology or from particular study results.

**Tab. 7.1:** Additional contributions of the study

| **Phase of research** | **Type of contribution** | **Discussion** |
|---|---|---|
| Explorative study | Methodology | The exploratory study was based on data gained from an e-learning platform used by second semester bachelor students. In the research community, there is a debate whether it is appropriate to use students as subjects for experimental studies.[766] Yet, the student datasets were helpful for analyses and exploration. Most important, the exploration based on the dataset led to the central idea of the theory, that is, in order to achieve acceptable estimation accuracy, the project members must use estimates as targets. Software professionals could corroborate the developed theory, based on this central idea. This gives another confirmation that students can be used as subjects for experimental as well as explorative studies. |
| | Methodology | The concept of simple random samples could be successfully adopted to simulate small development projects.[767] This approach offered a fast and easy-to-use instrument to assess the influence of different settings on estimation accuracy. |
| | Results | The analyses of the student datasets confirmed the individual differences paradigm.[768] The students were considerably different in any dimension addressed. They showed different solution capabilities, different solution efforts, as well as different working speeds.[769] Although students were classified in terms of capability, it was impossible to produce accurate estimates. It was necessary to incorporate project control and the PARKINSON effect, which both influence the actual effort, in order to get "good" estimates.[770] |
| Theory construction | Methodology | This study followed the theory classification scheme, recently recommended by GREGOR.[771] In particular, the EFT is an EP theory, which offers both explanation and prediction. In this context, this study gives a contribution by demonstrating how this classification scheme can be applied for theory construction. It gives an example of how to combine and present a causal model, assumptions, as well as propositions.[772] |

766 See introduction of Chapter 4.
767 See Section 4.3.4.
768 See Section 3.1.3.7.
769 See Sections 4.3.1, 4.3.2, and 4.3.3.
770 See Section 4.3.5.
771 GREGOR (2006). See Section 1.3.
772 See Sections 5.1, 5.2.2, and 5.2.3.

| Phase of research | Type of contribution | Discussion |
|---|---|---|
| Empirical testing | Methodology | This study shows how to test a theory by qualitative, semi-structured expert interviews in the domain of software development and software effort estimation. It gives reasons why theory construction is not contradictory to qualitative methods and an interpretive research position.[773] |
| | Methodology | This study made use of an expert review.[774] This approach is regarded as helpful in order to test whether incorrect conclusions have been drawn, whether the theory is comprehensible for practitioners, and whether the theory might have mistakenly turned into an ivory-tower theory. |
| | Results | Besides corroborating the EFT, the expert interview results provide a general insight into software development projects.[775] The responses show that the addressed software companies have many similarities and only a few differences in their process designs as well as estimation approaches. All companies have moved away from static, sequential development in favor of iterative development processes. Currently, agile development methodologies have a perceivable influence on software companies. |

## 7.3 Implications for Research and Practice

Projects that are affected by estimation fulfillment (EF-type) are supposed to perform differently than projects for which effort estimates and actual project efforts are independent figures (I-type). If this holds true, there are three important implications for research and practice, which are briefly discussed in the following.

### 7.3.1 Misinterpreted Research Results

Problems and challenges in software development projects might be overrated or misinterpreted. Research studies, which aim at analyzing the impact of particular factors on project performance, might be biased if simultaneously analyzing EF-type and I-type projects. EF-type projects might be completed with effort overruns of 25% on average without having faced any noteworthy problems during the development processes. A statistical analysis might mistakenly identify factors that appear responsible for effort overruns.

This idea is briefly illustrated by an example. Assuming a dataset used for a research study contains a number of EF-type projects. These projects have in common that they favor Java and Scrum. All EF-type projects show effort overruns around 25%. The I-type projects in the dataset ended with effort overruns of 30% as well as effort

[773] See Section 1.3.
[774] See Section 6.4.
[775] See Sections 6.2 and 6.3 as well as Appendix A.4.

underruns of 20%. Researchers might mistakenly assume that all projects face difficulties during development. A statistical analysis might incorrectly identify Scrum, for example, as a negative influence on project performance. Additionally, due to the difficulty of measuring productivity (see next section), the study might overlook that all EF-type projects perform on a higher productivity level than the I-type projects.

In order to prevent such incorrect conclusions, researchers must analyze the estimation fingerprint of an organization before turning to their central research questions. If a company shows a typical estimation fulfillment fingerprint, both the researchers and practitioners can identify the gap between effort estimates and average actual effort. This gap, e.g., 20% on average, must be excluded from the actual efforts in order to free the data from the estimation fulfillment effect.

### 7.3.2 Immeasurability of Productivity

The evolution of developer productivity over time might be immeasurable. Expertise-based estimation techniques, e.g., estimation by analogy, usually generate time-based estimates. Such techniques do not aim at metrics like Lines of Code or Function Points that reflect the software size. Therefore, the software size is unknown. However, the software size is necessary to quantify the output of a software development project. In consequence, it is not possible to immediately analyze the relationship of input, i.e., the actual effort, and the output.

In order to analyze productivity over time, researches (or practitioners) must generate software size metrics retrospectively. However, such figures might lead to incorrect study results.[776] Lines of Code vary with programming languages and the programming styles of different developers. Similarly, the ex-post generation of Function Points might also be biased. First, the known actual project efforts might influence the researcher. Second, the researcher might have insufficient knowledge and experience with the FPA. Third, Function Point metrics might be too low if functional specifications were incomplete at project start. This might be the case if developers use the effort estimates for interpreting vague and incomplete specifications, and if they implement more functionality than officially specified. In this case, the ratio of implemented Function Points per man-day would be too low. Forth, criticism on Function Points has noted that technology independence is not given. Fifth, Function Points might ignore the intended level of software quality and customer satisfaction,

[776] General problems of longitudinal analyses on developer productivity are addressed in Section 3.1.3.6.

which was achieved by tolerated gold-plating activities throughout the projects, not by quality specifications.

Additionally, if an EF-type company mistakenly responses to effort overruns by systematically increasing estimates by 20% in order to reduce the gap between actual and estimated effort, developers might still conform to the estimate, for example, by implementing more functionality than specified. In this case, a productivity analysis might incorrectly identify a negative evolution of developer productivity.

In general, companies that face estimation fulfillment are advised to refrain from simply increasing effort estimates in order to reduce the estimation gap. An increase of estimates might only lead to higher actual efforts without improving estimation accuracy.

### 7.3.3 Impact on Estimation Accuracy

EF-type projects and companies can bias estimation accuracy studies. As explained before, statistical analyses might mistakenly lead to the conclusion that one estimation technique is more accurate than another. Studies on estimation accuracy, for example, showed higher estimation accuracy for estimation by analogy in contrast to algorithmic approaches.[777] Such results must be controversially discussed when considering estimation fulfillment.

Algorithmic approaches often require detailed information. Accordingly, project management might have already written detailed specifications before turning to effort estimation. In this case, detailed specifications might influence developers how to interpret work packages while they emancipate from the given estimates. In consequence, potential effort overruns are not based on estimation fulfillment, but they actually refer to inaccurate estimates. If a dataset contains both EF-type and I-type projects, this aspect must be taken into account and be eliminated before comparing estimation techniques.

## 7.4 Limitations

Although this study was planned, designed, conducted, and reviewed as rigorously as possible, it has – like any other research – various limitations:

- The presented understanding of the nature of software development might not be shared by all members of the research community.

[777] See Section 2.4.

While it was attempted to present the underlying understanding thoroughly, someone might disagree with it, and, in consequence, disagree with the theory's foundation.

- The exploratory study used second semester students as subjects. Moreover, it was based on simulated development projects, which, in turn, were based on exercises solution attempts. It might be debatable whether this approach is acceptable or not.

- The theory construction focused on explanation and prediction. The theory could have addressed the goal of *"design and action"* as well.[778] This would contribute to the relevance of this thesis. For example, the theory could have given advice to project managers on how to react if they are confronted with estimation fulfillment.

- The theory was validated on the basis of semi-structured expert interviews. Typical difficulties, problems, and pitfalls in using qualitative interviews have been addressed in Section 6.1.1. Even though much effort was invested in the design of the interview script, and although much attention was given during the expert interviews not to make such mistakes, the conducted interviews might still have been biased, especially with regard to the *construction of knowledge* and the *ambiguity of language.*

- The selection of experts limits this study. Although the interviewees look back to hundreds of development projects referring to different industry sectors, any further interview could provide an interesting counter-finding or even falsify the theory. Particularly, all companies are located in Germany or Switzerland. Therefore, this study ignores (at least) culture as a potential building block of the theory's causal model. In addition, there might be special industry sectors for which the theory does not apply.

- As a final point, the combination of interpretive and positivist understandings might face opposition of those who strictly regard these understandings as contrary, conflicting, and/or being mutually exclusive.

[778] GREGOR (2006). See Tab. 1.1.

## 7.5 Outlook

The goal of this thesis was to analyze the relationship of estimated and actual development effort. Thus, a theory was developed in order to explain how and why estimates have an essential impact on actual project efforts. Besides, this thesis gives name to a complex phenomenon. This (at least) might inspire other researchers to explore the field of estimation fulfillment. There are various questions and aspects touched, but left open by this study:

- The (optional) proposition P3 was stated as part of the EFT, but it was excluded from the theory validation.[779] This proposition can be subject to future research. However, such an in-depth study of software development projects is – if at all feasible – regarded as difficult and time-consuming.
- The estimation fingerprints are a suggestion to identify different types of relationships between estimated and actual development effort. Future research can confront the concept of estimation fingerprints with quantitative empirical project data.
- Another interesting aspect is to explore in more detail *why* some companies are affected by estimation fulfillment while others are not. This would contribute to and fine-tune the set of stated assumptions.
- Similarly, it would be interesting to see whether the EFT still holds true when being applied in different cultural settings. This could be analyzed by conducting expert interviews with software professionals from U.S. companies or Asian companies, e.g., from India, China, or Japan.
- As a final point, research can address the perspective of top management on estimation fulfillment. It would be interesting to know if there is a general concern, or if top managers ignore the phenomenon as long as profits are made.

[779] See Section 5.2.3. Proposition 3: *If individual differences between developers are systematically and precisely documented and taken appropriately into account during effort estimation, the deviation of estimated and actual effort will reduce for EF-type projects.*

This research is just a small step towards a broad understanding of self-fulfilling estimates. More research is undoubtedly needed to explore and understand the effect of estimation fulfillment in detail.

# REFERENCES

Abdel-Hamid, T.; Madnick, S. E.: *Software Project Dynamics: An Integrated Approach*. Prentice Hall, 1991.

Abdel-Hamid, T. K.; Madnick, S. E.: *Special Feature: Impact of Schedule Estimation on Software Project Behavior*. In: IEEE Software, 3 (1986) 4, pp. 70-75.

Abran, A.; Robillard, P. N.: *Function points analysis: an empirical study of its measurement processes*. In: Software Engineering, IEEE Transactions on, 22 (1996) 12, pp. 895-910.

Agarwal, R.; Prasad, J.: *Are Individual Differences Germane to the Acceptance of New Information Technologies?* In: Decision Sciences, 30 (1999) 2, pp. 361-391.

Agrawal, M.; Chari, K.: *Software Effort, Quality, and Cycle Time: A Study of CMM Level 5 Projects*. In: Software Engineering, IEEE Transactions on, 33 (2007) 3, pp. 145-156.

Akintoye, A.: *Analysis of factors influencing project cost estimating practice*. In: Construction Management & Economics, 18 (2000) 1, pp. 77-89.

Albrecht, A. J.: *Measuring Application Development Productivity*. In: Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium. Monterey, California 1979, pp. 83–92.

Albrecht, A. J.; Gaffney Jr, J. E.: *Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation*. In: IEEE Transactions on Software Engineering, 9 Special Edition (1983) 6, pp. 639-648.

American Society of Civil Engineers: *Journal of Construction Engineering and Management*. 2008. Available at http://scitation.aip.org/coo/. Last access on.

Arditi, D.; Mochtar, K.: *Trends in productivity improvement in the US construction industry*. In: Construction Management & Economics, 18 (2000) 1, pp. 15-27.

Armstrong, W. W.: *Dependency structures of database relationships*. In: IFIP Congress, 74 (1974), pp. 580–583.

Bach, J.: *Enough about process: what we need are heroes*. In: Computer, 12 (1995) 2, pp. 96-98.

Bach, J.: *What Software Reality Is Really About*. In: Computer, 32 (1999) 12, pp. 148-149.

Baloff, N.: *Startups in Machine-lntensive Production Systems*. In: Journal of Industrial Engineering, 17 (1966) 1, pp. 25-32.

Baloff, N.: *Extension of the Learning Curve-Some Empirical Results*. In: Operations Research Quarterly, 22 (1971) 4, pp. 329-340.

Balzert, H.: *UML Kompakt*. 2. Edition. Spektrum Akad. Verlag, München, 2005.

Banker, R. D.; Chang, H.; Kemerer, C. F.: *Evidence on Economies of Scale in Software Development*. In: Information and Software Technology, 36 (1994) 5, pp. 275-282.

Banker, R. D.; Kauffman, R. J.; Kumar, R.: *An Empirical Test of Object-Based Output Measurement Metrics in a Computer Aided Software Engineering (CASE) Environment*. In: Journal of Management Information Systems, 8 (1991) 3, pp. 127-150.

Banker, R. D.; Kemerer, C. F.: *Scale Economies in New Software Development*. In: IEEE Transactions on Software Engineering, 15 (1989) 10, pp. 1199-1205.

Barnett, L.: *Outliers in Statistical Data*. 3. Edition. Wiley, 1994.

Bartol, K. M.; Martin, D. C.: *Managing Information Systems Personnel: A Review of Literature and Managerial Implications*. In: MIS Quarterly, 6 (1982) 1, pp. 49-70.

Basili, V. R.; Turner, A. J.: *Iterative Enhancement: A Practical Technique for Software Development*. In: IEEE Transactions on Software Engineering, (1975), pp. 390-396.

Beaver, J. M.; Schiavone, G. A.: *The Effects of Development Team Skill on Software Product Quality*. In: ACM SIGSOFT Software Engineering Notes, 31 (2006) 3, pp. 1-5.

Beck, K.; Andres, C.: *Extreme Programming Explained: Embrace Change*. 2. Edition. Addison-Wesley Professional, 2004.

Beck, K.; Beedle, M.; van Bennekum, A.; Cockburn, A.; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J.; Hunt, A.; Jeffries, R.; Kern, J.; Marick, B.; Martin, R. C.; Mellor, S.; Schwaber, K.; Sutherland, J.; Thomas, D.: *The Agile Manifesto*. 2001. Available at http://www.agilemanifesto.org/. Last access on 2008-09-20.

Becker, J.; Niehaves, B.: *Epistemological perspectives on IS research: a framework for analysing and systematizing epistemological assumptions*. In: Information Systems Journal, 17 (2007), pp. 197-214.

Beecham, S.; Baddoo, N.; Hall, T.; Robinson, H.; Sharp, H.: *Motivation in Software Engineering: A systematic literature review*. In: Information and Software Technology, 50 (2008) 9-10, pp. 860-878.

Beeri, C.; Bernstein, P. A.: *Computational problems related to the design of normal form relational schemas*. In: ACM Transactions on Database Systems (TODS), 4 (1979) 1, pp. 30-59.

Belady, L. A.; Lehman, M. M.: *A Model of Large Program Development*. In: IBM System Journal, 15 (1976) 3, pp. 225-252.

Belady, L. A.; Lehman, M. M.: *Program System Dynamics or the Metadynamics of Systems in Maintenance and Growth*. In: Program Evolution. Edited by M. M. Lehman, L. A. Belady. Academic Press, London, 1985.

Bergeron, F.; St-Arnaud, J. Y.: *Estimation of information systems development efforts: a pilot study*. In: Information & Management, 22 (1992) 4, pp. 239-254.

Berry, D. M.: *The Inevitable Pain of Software Development: Why There Is No Silver Bullet*. In: Proceedings of the Radical Innovations of Software and Systems Engineering in the Future. Eds.: M. Wirsing, A. Knapp, S. Balsamo. Venice, Italy 2004, pp. 50-74.

Biggeleben, M.: *From Communication to Implementation*. In: Proceedings of the 8. Internationale Tagung Wirtschaftsinformatik. Karlsruhe 2007.

Biggeleben, M.; Grgecic, D.; Holten, R.; Schäfermeyer, M.: *E-Learning-Szenarien an der Massenuniversität - Technische Realisierung und Erfolgsmessung*. In: E-Learning in Hochschule und Weiterbildung. Einsatzchancen und Erfahrungen. Reihe: Erwachsenenbildung und lebenslanges Lernen. Edited by R. Holten, D. Nittel. Frankfurt, 2009a, S. 127-145.

Biggeleben, M.; Holten, R.: *Enlist*. 2009a. Available at http://www.ise.wiwi.uni-frankfurt.de/enlist/frontend/index.php. Last access on 2009-08-10.

Biggeleben, M.; Holten, R.: *Spielwiesen/Playgrounds*. 2009b. Available at http://www.ise.wiwi.uni-frankfurt.de/spielwiesen/. Last access on 2009-08-10.

Biggeleben, M.; Kolbe, H.; Schäfermeyer, M.; Vranesic, H.: *Prüfkriterien für Geschäftsmodelle im Kontext von Software as a Service*. In: Proceedings of the 9. Internaltionale Tagung Wirtschaftsinformatik. Vienna 2009b.

Blanche, M. T.; Durrheim, K.; Painter, D.: *Research in Practice: Applied Methods for the Social Sciences*. 2. Edition. UTC Press, 2008.

Bless, C.; Higson-Smith, C.; Kagee, A.: *Fundamentals of Social Research Methods: An African Perspective*. 4. Edition. Juta Legal and Academic Publishers, 2007.

Boehm, B.: *A view of 20th and 21st century software engineering*. In: Proceedings of the 28th International Conference on Software Engineering. Shanghai, China 2006, pp. 12-29.

Boehm, B.; Abts, C.; Chulani, S.: *Software development cost estimation approaches - A survey*. In: Annals of Software Engineering, 10 (2000a) 1, pp. 177-205.

Boehm, B.; Basili, V. R.: *Gaining Intellectual Control of Software Development*. In: Computer, 33 (2000) 5, pp. 27-33.

Boehm, B.; Basili, V. R.: *Software Defect Reduction Top 10 List*. In: Computer, 34 (2001) 1, pp. 135-137.

Boehm, B.; Turner, R.: *Using Risk to Balance Agile and Plan-Driven Methods*. In: Computer, 36 (2003) 6, pp. 57-66.

Boehm, B. W.: *The High Cost of Software*. In: Practical Strategies for Developing Large Software Systems. Edited by E. Horowitz. Addison-Wesley, Reading, MA, 1975.

Boehm, B. W.: *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, 1981.

Boehm, B. W.: *A Spiral Model of Software Development and Enhancement*. In: Proceedings of the International Workshop on Software Process and Software Environments. 1985.

Boehm, B. W.: *Industrial software metrics top 10 list*. In: IEEE Software, 4 (1987) 5, pp. 84-85.

Boehm, B. W.: *A Spiral Model of Software Development and Enhancement*. In: IEEE Computer, 21 (1988) 4, pp. 61-72.

Boehm, B. W.; Abts, C.; Brown, A. W.; Chulani, S.; Clark, B. K.; Horowitz, E.; Madachy, R.; Reifer, D. J.; Steece, B.: *Cost Estimation with COCOMO II*. Prentice-Hall, 2000b.

Boehm, B. W.; Clark, B.; Horowitz, E.; Westland, C.; Madachy, R.; Selby, R.: *Cost Models for Future Software Life Cycle Processes: COCOMO II*. In: Annals of Software Engineering, Special Volume on Software Process and Product Measurement (1995), pp. 45-60.

Bortz, J.; Döring, N.: *Forschungsmethoden und Evaluation für Human-und Sozialwissenschaftler: Für Human-und Sozialwissenschaftler*. 4. Edition. Springer, Berlin, 2006.

Breakwell, G. M.; Smith, J. A.; Fife-Schaw, C.; Hammond, S.: *Research methods in psychology*. 3. Edition. Sage, 2006.

Briand, L. C.; Emam, K. E.; Surmann, D.; Wieczorek, I.; Maxwell, K. D.: *An assessment and comparison of common software cost estimation modeling techniques*. In: Proceedings of the 21st international conference on Software engineering. Los Angeles 1999, pp. 313-322

Brooks, F. P.: *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, Reading, MA, 1975.

Brooks, F. P.: *No Silver Bullet: Essence and Accidents of Software Engineering*. In: IEEE Computer, 20 (1987) 4, pp. 10-19.

Brooks, F. P.: *The Mythical Man-Month after 20 Years*. 1995a. Available at ftp.cs.washington.edu/pub/se/icse17/brooks.ps.Z. Last access on 2008-09-21.

Brooks, F. P.: *The Mythical Man-Month: Essays on Software Engineering*. Anniversary Edition. Addison-Wesley, Reading, MA, 1995b.

Brown, P. S.; Gould, J. D.: *An experimental study of people creating spreadsheets*. In: ACM Transactions on Information Systems (TOIS), 5 (1987) 3, pp. 258-272.

Bundschuh, M.; Dekkers, C.: *Variants of the IFPUG Function Point Counting Method*. In: The IT Measurement Compendium - Estimating and Benchmarking Success with Functional Size Measurement Edited, Springer Berlin, Heidelberg, 2008, pp. 397-407.

Burrell, G.; Morgan, G.: *Sociological Paradigms and Organisational Analysis*. Ashgate Publishing, Aldershot, UK et al., 1979.

Carlson, J. G.: *Cubic Learning Curves: Precision Tool for Labor Estimating*. In: Manufacturing Engineering and Management, 71 (1973) 5, pp. 22-25.

Carlson, J. G.: *How Much Does Forgetting Cost?* In: Industrial Engineering, 8 (1976) 9, pp. 40-47.

Chamberlin, D. D.; Boyce, R. F.: *SEQUEL: A Structured English Query Language*. In: Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description. 1974, pp. 249–264.

Chapman, A.: *Conscious Competence Learning Model*. 2009. Available at http://www.businessballs.com/consciouscompetencelearningmodel.htm. Last access on 2009-02-19.

Chatman, V. V.: *CHANGE-POINTs: A proposal for software productivity measurement*. In: Journal of Systems and Software, 31 (1995) 1, pp. 71-91.

Cheesman, J.; Daniels, J.: *UML components: a simple process for specifying component-based software*. Addison-Wesley, 2000.

Chen, W. S.; Hirschheim, R.: *A paradigmatic and methodological examination of information systems research from 1991 to 2001*. In: Information Systems Journal, 14 (2004) 3, pp. 197-235.

Chidamber, S. R.; Kemerer, C. F.: *A Metrics Suite for Object Oriented Design*. In: IEEE Transactions on Software Engineering, 20 (1994) 6, pp. 476-493.

Chulani, S.: *Bayesian Analysis of Software Cost and Quality Models (Dissertation)*. University of Southern California, Los Angeles, 1999.

Chulani, S.; Boehm, B.; Steece, B.: *Bayesian Analysis of Empirical Software Engineering Cost Models*. In: IEEE Transactions on Software Engineering, 25 (1999) 4.

Clark, B.; Devnani-Chulani, S.; Boehm, B.: *Calibrating the COCOMO II Post-Architecture Model*. In: Proceedings of the 20th International Conference on Software Engineering. Kyoto, Japan 1998, pp. 477-480.

Clarkson, M.: *Developing It Staff: A Practical Approach*. Springer, 2001.

Cochran, E. B.: *New concepts of the Learning Curve*. In: Journal of Industrial Engineering, 11 (1960) 4, pp. 317-327.

Cockburn, A.; Highsmith, J.: *Agile Software Development: The People Factor*. In: IEEE Computer, 34 (2001) 11, pp. 131-133.

Codd, E. F.: *A Relational Model of Data for Large Shared Data Banks*. In: Communications of the ACM, 13 (1970) 6, pp. 377-387

Codd, E. F.: *Further Normalization of the Database Relational Model*. In: Data Base Systems. Edited by R. Rustin. Prentice-Hall, Englewood Cliffs, NJ, 1972.

Codd, E. F.: *Recent Investigations into Relational Data Base Systems*. In: Proceedings of the Information Processing 74. Amsterdam 1974, pp. 1017–1021.

Cohen, J.: *Statistical Power Analysis for the Behavioral Sciences*. 2. Edition. 1988.

Cohn, M.: *Agile Estimating and Planning*. Prentice Hall, 2005.

Cohn, M.; Ford, D.: *Introducing an Agile Process to an Organization*. In: Computer, 36 (2003) 6, pp. 74-78.

Conte, S. D.; Dunsmore, H. E.; Shen, V. Y.: *Software Engineering Metrics and Models*. Benjamin-Cummings Publishing Co. Inc., Redwood City, CA, 1986.

Conway, R.; Schultz, A.: *The Manufacturing Progress Function*. In: Journul of Industrial Engineering, 10 (1959) 1, pp. 39-53.

Cooper, A. C.: *R&D Is More Efficient in Small Companies*. In: Harvard Business Review, 42 (1964) 3, pp. 75-83.

Coplien, J. O.: *Multi-Paradigm Design*. 1999.

Coram, M.; Bohner, S.: *The impact of agile methods on software project management*. In: Proceedings of the Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops on the. 2005, pp. 363-370.

Corsten, H.: *Produktionswirtschaft: Einfuehrung in das industrielle Produktionsmanagement*. 11 Edition. Oldenbourg, München, 2007.

COSMIC: *The COSMIC Functional Size Measurement Method Version 3.0 - Measurement Manual*. 2007a. Available at http://www.gelog.etsmtl.ca/cosmic-ffp/updates/COSMIC_Method_v3.0_Measurement_Manual.pdf. Last access on 2009-01-14.

COSMIC: *The COSMIC Functional Size Measurement Method Version 3.0 - Method Overview*. 2007b. Available at http://www.gelog.etsmtl.ca/cosmic-ffp/updates/COSMIC_Method_v3.0_Documentation_Overview.pdf. Last access on 2009-01-13.

COSMIC: *History of Functional Size Measurement*. 2008. Available at http://www.cosmicon.com/historycs.asp. Last access on 2009-01-13.

COSMIC: *Overview of the COSMIC FFP Method*. 2009. Available at http://www.cosmicon.com/overview.asp. Last access on 2009-01-14.

Cox, B. J.: *Planning the software industrial revolution*. In: IEEE Software, 7 (1990) 6, pp. 25-33.

Crosby, P. B.: *Quality Is Free*. McGraw-Hill, 1979.

Curtis, B.: *Fifteen years of psychology in software engineering: Individual differences and cognitive science*. In: Proceedings of the 7th international Conference on Software Engineering. 1984, pp. 97-106

Custer, R. L.; Scarcella, J. A.; Stewart, B. R.: *The Modified Delphi Technique - A Rotational Modification*. In: Journal of Vocational and Technical Education, 15 (1999) 2, pp. 50-58.

D'Souza, D. F.; Wills, A. C.: *Objects, components, and frameworks with UML*. Addison-Wesley Reading, MA, 1998.

Dalkey, N.; Helmer, O.: *An Experimental Application of the Delphi Method to the Use of Experts*. In: management Science, 9 (1963) 3, pp. 458-467.

Davidson, F. P.; Huot, J. C.: *Management trends for major projects*. In: Project Appraisal, 4 (1989) 3, pp. 133–142.

Davis, A. M.: *Fifteen Principles of Software Engineering*. In: IEEE Software, 11 (1994) 6, pp. 94-96, 101.

DeMarco, T.; Lister, T. R.: *Peopleware*. Dorset House Pub. Co New York, NY, New York, NY, 1987.

Dickey, T. E.: *Programmer variability*. In: Proceedings of the IEEE, 69 (1981) 7, pp. 844-845.

Dickmann, M. H.; Stanford-Blair, N.: *Connecting Leadership to the Brain*. Corwin Press, 2000.

Dijkstra, E. W.: *Letters to the editor: go to statement considered harmful*. In: Communications of the ACM, 11 (1968) 3, pp. 147-148.

Dijkstra, E. W.: *The humble programmer*. In: Communications of the ACM, 15 (1972) 10, pp. 859-866.

Dolado, J. J.: *A study of the relationships among Albrecht and Mark II Function Points, lines of code 4GL and effort*. In: Journal of Systems and Software, 37 (1997) 2, pp. 161-173.

Donnelly, R. A.: *The complete idiot's guide to statistics*. Alpha, 2004.

Douglas McIlroy, M.: *Mass-Produced Software Components*. In: Software Engineering Concepts and Techniques (1968 NATO Conference of Software Engineering). Edited by J. M. Buxton, P. Naur, B. Randell. NATO Science Committee, 1968, pp. 79-87.

Dunning, D.: *Strangers to ourselves*. In: The Psychologist, 19 (2006) 10, pp. 600-603.

Dunning, D.; Johnson, K.; Ehrlinger, J.; Kruger, J.: *Why people fail to recognize their own incompetence*. In: Current Directions in Psychological Science, 12 (2003) 3, pp. 83-87.

Dunning, D.; Kruger, J.: *Unskilled and Unaware of It: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments*. In: Journal of Personality and Social Psychology, 77 (1999) 6, pp. 1121–1134.

Ebbinghaus, H.: *Memory: A contribution to experimental psychology (Translation published in 1964)*. 1885.

Edmonds, E. A.: *A process for the development of software for non-technical users as an adaptive system*. In: General Systems: Yearbook of the Society for General Systems Research, 19 (1974), pp. 215–218.

Efron, B.: *Bootstrap Methods: Another Look at the Jackknife*. In: The Annals of Statistics, 7 (1979a) 1, pp. 1-26.

Efron, B.: *Computers and the Theory of Statistics: Thinking the Unthinkable*. In: SIAM Review, 21 (1979b) 4, pp. 460-480.

Efron, B.; Tibshirani, R.: *An Introduction to the Bootstrap*. Chapman and Hall, New York, 1993.

Fagan, M. E.: *Design and code inspections to reduce errors in program development*. In: IBM System Journal, 15 (1976) 3, pp. 182-211.

Falconer, D. J.; Mackay, D. R.: *Ontological Problems of Pluralist Research Methodologies* Eds.: W. D. Haseman, D. L. Nazareth, D. Goodhue. Milwaukee 1999, pp. 624-626.

Fandel, G.; Blaga, S.: *Aktivitätsanalytische Überlegungen zu einer Theorie der Dienstleistungsproduktion*. In: Zeitschrift für Betriebswirtschaft, 1 (2004) 2004, pp. 1-21.

Finnie, G. R.; Wittig, G. E.; Desharnais, J. M.: *A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models*. In: Journal of Systems and Software, 39 (1997) 3, pp. 281-289.

Fitzgerald, B.; Howcroft, D.: *Competing dichotomies in IS research and possible strategies for resolution*. In: Proceedings of the 19th International Conference on Information Systems (ICIS). Helsinki, Finland 1998.

Fitzgerald, G.; Hirschheim, R.; Mumford, E.; Wood-Harper, A. T.: *Information Research Methodology: An Introduction to the Debate*. In: Research Methods in Information Systems. Edited by R. Hirschheim, G. Fitzgerald, E. Mumford, A. T. Wood-Harper. Amsterdam, 1985, pp. 3-9.

Flyvbjerg, B.; Holm, M. S.; Buhl, S.: *Underestimating Costs in Public Works Projects*. In: Journal of the American Planning Association, 68 (2002) 3, pp. 279-295.

Fontana, A.; Frey, J. H.: *The interview: From structured questions to negotiated text*. In: Handbook of qualitative research. Edited by Y. S. Lincoln, N. K. Denzin. Sage, Thousand Oaks, CA, 2000, pp. 645-672.

Forrester, J. W.: *Industrial dynamics*. Pegasus Communications, Waltham, MA, 1961.

Forrester, J. W.: *Principles of Systems*. 2nd Edition. Pegasus Communications, Waltham, MA, 1968.

Forrester, J. W.: *Urban Dynamics*. Pegasus Communications, Waltham, MA, 1969.

Fowler, M.: *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3. Edition. Addison-Wesley, Boston, MA, 2003.

Fraser, S.; Mancl, D.: *No Silver Bullet: Software Engineering Reloaded*. In: IEEE Software, 25 (2008) 1, pp. 91-94.

Fraser, S. D.; Brooks, F. P.; Fowler, M.; Lopez, R.; Namioka, A.; Northrop, L.; Parnas, D. L.; Thomas, D.: *"No silver bullet" reloaded: Retrospective on "Essence and Accidents of Software Engineering"*. Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion. Montreal, Quebec, Canada 2007.

Frohnhoff, S.: *Große Softwareprojekte*. In: Informatik-Spektrum, 31 (2008) 6, pp. 566-575.

Fuchs, T.: *script.aculo.us - web 2.0 javascript*. 2009. Available at http://script.aculo.us/. Last access on 2009-06-12.

Funahashi, K.: *On the approximate realization of continuous mappings by neural networks*. In: Neural Networks, 2 (1989) 3, pp. 183-192

Galea, S.: *The Boeing Company: 3D Function Point Extensions, V2.0, Release 1.0*. Boeing Information and Support Services, Research and Technology Software Engineering, 1995.

Garmus, D.; Herron, D.: *Function Point Analysis: Measurement Practices for successful Software Projects*. Addison-Wesley, Boston, MA, 2001.

Garnett, N.; Pickrell, S.: *Benchmarking for construction: theory and practice*. In: Construction Management & Economics, 18 (2000) 1, pp. 55-63.

Gibbs, W. W.: *Software's chronic crisis*. In: Scientific American, 271 (1994) 3, pp. 72-81.

Gibson, M.; Arnott, D.: *The Use of Focus Groups in Design Science Research*. In: Proceedings of the 18th Australasian Conference on Information Systems. Toowoomba 2007, pp. 327-337.

Gielecki, M.; Hewlett, J.: *Commercial Nuclear Electric Power in the United States: Problems and Prospects*. 1994.

Glass, R. L.: *Facts and Fallacies of Software Engineering*. Addison-Wesley Professional, 2002.

Glass, R. L.: *IT Failure Rates - 70% or 10-15%?* In: IEEE Software, 22 (2005) 3, pp. 112, 110-111.

Glass, R. L.: *The Standish report: does it really describe a software crisis?* In: Communications of the ACM, 49 (2006) 8, pp. 15-16.

Goel, A. L.: *Software reliability models: Assumptions, limitations, and applicability*. In: IEEE Transactions on Software Engineering, SE 11 (1985) 12, pp. 1411-1423.

Goldratt, E. M.: *Critical Chain*. North River Press, Great Barrington, MA, 1997.

Google: *Google Analytics*. 2009. Available at http://www.google.com/intl/en/analytics/index.html. Last access on 2009-07-29.

Grady, R. B.: *Practical software metrics for project management and process improvement*. Prentice Hall, New Yersey, 1992.

Greenfield, J.; Short, K.: *Software factories: assembling applications with patterns, models, frameworks and tools*. In: Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. 2003, pp. 16-27.

Gregor, S.: *The Nature of Theory in Information Systems*. In: MIS Quarterly, 30 (2006) 3, pp. 611-642.

Halstead, M. H.: *Elements of Software Science*. Elsevier, New York, 1977.

Harrison, W.: *Clueless-and oblivious*. In: IEEE Software, 21 (2004) 3, pp. 5-7.

Harter, D. E.; Krishnan, M. S.; Slaughter, S. A.: *Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development*. In: management Science, 46 (2000) 4, pp. 451-466.

Harvey, N.: *Improving judgment in forecasting*. In: Principles of Forecasting: A Handbook for Researchers and Practitioners. Edited by J. S. Armstrong. Boston, 2001, pp. 59-80.

Hatch, M. J.; Cunliffe, A. L.: *Organization Theory: Modern, Symbolic, and Postmodern Perspectives*. 2. Edition. Oxford University Press, New York, NY, USA, 2006.

Hayes, N.; Walsham, G.: *Participation in groupware-mediated communities of practice: a socio-political analysis of knowledge working*. In: Information and Organization, 11 (2001) 4, pp. 263-288.

Haykin, S.: *Neural Networks: A Comprehensive Foundation*. 2. Edition. Prentice Hall, 1998.

He, J.; Butler, B. S.; King, W. R.: *Team Cognition: Development and Evolution in Software Project Teams*. In: Journal of Management Information Systems, 24 (2007) 2, pp. 261-292.

Heemstra, F. J.: *Software cost estimation*. In: Information and Software Technology, 34 (1992) 10, pp. 627-639.

Heinemeier Hansson, D.: *Ruby on Rails*. 2008. Available at http://www.rubyonrails.org/. Last access on 2009-06-12.

Hetzel, B.: *Making Software Measurement Work: Building an Effective Measurement Program*. John Wiley & Sons, Inc., New York, NY, 1993.

Hevner, A. R.; March, S. T.; Park, J.; Ram, S.: *Design Science in Information Systems Research*. In: MIS Quarterly, 28 (2004) 1, pp. 75-105.

Highsmith, J.: *Agile Software Development Ecosystems*. Addison-Wesley Longman, Amsterdam, 2002.

Highsmith, J.; Cockburn, A.: *Agile Software Development: The Business of Innovation*. In: IEEE Computer, 34 (2001) 9, pp. 120-127.

Hihn, J.; Habib-Agahi, H.: *Cost estimation of software intensive projects: a survey of current practices*. In: Proceedings of the 13th International Conference on Software Engineering. Austin, Texas, USA 1991, pp. 276–287.

Hill, J.; Thomas, L. C.; Allen, D. E.: *Experts' estimates of task durations in software development projects*. In: International Journal of Project Management, 18 (2000) 1, pp. 13-21.

Hill, T. P.: *On Goods and Services*. In: Review of Income and Wealth, 23 (1977) 4, pp. 315-338.

Hirschheim, R.: *Information Systems Epistemology: An Historical Perspective*. In: Research Methods in Information Systems. Edited by E. Mumford, R. Hirschheim, G. Fitzgerald, A. T. Wood-Harper. Amsterdam, 1985, pp. 13-35.

Hollmann, J. K.; Dysert, L. R.: *Escalation Estimation: Working With Economics Consultants*. In: AACE International Transactions, (2007), pp. 01.01-01.06.

Holten, R.; Dreiling, A.; Becker, J.: *Ontology-Driven Method Engineering for Information Systems Development*. In: Ontological Analysis, Evaluation, and Engineering of Business Systems Analysis Methods. Edited by P. Green, M. Rosemann. IDEA Group Publishing, Hershey, 2004, pp. 174-215.

Howard, A.: *On site: Software Engineering Project Management*. In: Communications of the ACM, 44 (2001) 5, pp. 23-24.

Howells, J.: *Innovation, Consumption and Services: Encapsulation and the Combinatorial Role of Services*. In: The Service Industries Journal, 24 (2004) 1, pp. 19-36.

Humphrey, W. S.: *Managing the Software Process*. Addison-Wesley, 1989.

Hunt, A.; Thomas, D.: *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 2000.

IEEE Computer Society: *The Software Engineering Body of Knowledge - Chapter 11*. 2008a. Available at http://www2.computer.org/portal/web/swebok/html/ch11#INTRODUCTION. Last access on 2009-06-10.

IEEE Computer Society: *The Software Engineering Body of Knowledge - Introduction to the Guide*. 2008b. Available at http://www2.computer.org/portal/web/swebok/html/ch1#Ref1. Last access on 2009-06-10.

IEEE Standards Association: *IEEE Standard Glossary of Software Engineering Terminology*. 1990.

InfoQ.com: *Interview: Jim Johnson of the Standish Group*. 2006. Available at http://www.infoq.com/articles/Interview-Johnson-Standish-CHAOS. Last access on 2008-09-30.

International Function Point Users Group: *Function Point Counting Practices Manual, Release 4.0*. Westerville, Ohio, 1994.

ISO: *ISO 9001:2008, Quality Management Systems - Requirements*. International Organization for Standardization, Geneva, Switzerland, 2008.

Jackson, M. A.: *The Role of Architecture in Requirements Engineering*. In: Proceedings of the Proceedings of the IEEE International Conference on Requirements Engineering. Colorado Springs 1994, pp. 241.

Jacobson, I.; Christensen, M.; Jonsson, P.; Overgaard, G.: *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, New York, 1992.

Jantzen, K.: *Verfahren der Aufwandsschätzung für komplexe Softwareprojekte von heute*. In: Informatik-Spektrum, 31 (2008) 1, pp. 35-49.

Jeffery, D. R.; Low, G. C.; Barnes, M.: *A Comparison of Function Point Counting Techniques*. In: IEEE Transactions on Software Engineering, 19 (1993) 5, pp. 529-532.

Jenkins, A. M.; Naumann, J. D.; Wetherbe, J. C.: *Empirical Investigation of Systems Development Practices and Results*. In: Information & Management, 7 (1984) 2, pp. 73-82.

Jones, C.: *A short History of Function Points and Feature Points*. Software Productivity Reserch Inc., 1988.

Jones, C.: *Applied Software Measurement: Assuring Productivity and Quality*. McGraw-Hill, New York, 1996.

Jones, C.: *Applied Software Measurement: Global Analysis of Productivity and Quality*. 3. Edition. McGraw-Hill Professional, 2008.

Jorgensen, M.: *A review of studies on expert estimation of software development effort*. In: Journal of Systems and Software, 70 (2004) 1-2, pp. 37-60.

Jorgensen, M.: *Estimation of Software Development Work Effort: Evidence on Expert Judgment and Formal Models*. In: International Journal of Forecasting, 23 (2007) 3, pp. 449-462.

Jorgensen, M.; Boehm, B.: *Software Development Effort Estimation: Formal Models or Expert Judgment?* In: IEEE Software, (2009).

Jorgensen, M.; Grimstad, S.: *Over-optimism in Software Development Projects: The Winner's Curse*. In: Proceedings of the International Conference on Electronics, Communications and Computers. 2005, pp. 280-285.

Jorgensen, M.; Molokken, K.: *How large are software cost overruns? A review of the 1994 CHAOS report*. In: Information and Software Technology, 48 (2006) 4, pp. 297-301.

Jorgensen, M.; Shepperd, M.: *A Systematic Review of Software Development Cost Estimation Studies*. In: IEEE Transactions on Software Engineering, 33 (2007) 1, pp. 33-53.

Jorgensen, M.; Sjoberg, D.: *Empirical studies on effort estimation in software development projects*. In: Proceedings of the International Conference on Challenges of Information Technology Management in the 21st Century. Anchorage, Alaska, United States 2000, pp. 778-779.

Jorgensen, M.; Sjoberg, D. I. K.: *Impact of effort estimates on software project work*. In: Information and Software Technology, 43 (2001) 15, pp. 939-948.

Kagel, J. H.; Levin, D.: *The Winner's Curse and Public Information in Common Value Auctions*. In: American Economic Review, 76 (1986) 5, pp. 894.

Kan, S. H.: *Metrics and Models in Software Quality Engineering*. Addison-Wesley, 1993.

Kaner, C.; Bach, J.; Pettichord, B.: *Lessons Learned in Software Testing: A Context-Driven Approach*. Wiley, 2002.

Karner, G.: *Metrics for Objectory*. Master Thesis, Linkoping University (LiTH-IDA-Ex-9344:21), Linkoping, Sweden, 1993.

Kauffman, R.; Kumar, R.: *Modeling Estimation Expertise in Object Based ICASE Environments*. Report. Stern School of Business, New York University. New York 1993.

Kemerer, C. F.: *An empirical validation of software cost estimation models*. In: Communications of the ACM, 30 (1987) 5, pp. 416-429.

Kemerer, C. F.: *Reliability of function points measurement: a field experiment*. In: Communications of the ACM, 36 (1993) 2, pp. 85-97.

Kernighan, B. W.; Ritchie, D. M.: *The C Programming Language*. Prentice Hall, 1978.

Kerzner, H.: *Project Management: A Systems Approach to Planning, Scheduling and Controlling*. 2. Edition. New York, 1984.

Khoshgoftaar, T.; Pandya, A.; Lanning, D.: *Application of neural networks for predicting program faults*. In: Annals of Software Engineering, 1 (1995) 1, pp. 141-154.

Kilian-Kehr, R.; Terzidis, O.; Voelz, D.: *Industrialization of the Software Sector*. In: Wirtschaftinformatik, 49 (2007) Special Issue, pp. 62-71.

Kitchenham, B.; Lawrence Pfleeger, S.; McColl, B.; Eagan, S.: *An empirical study of maintenance and development estimation accuracy*. In: Journal of Systems and Software, 64 (2002a) 1, pp. 57-77.

Kitchenham, B.; Linkman, S.: *Estimates, uncertainty, and risk*. In: IEEE Software, 14 (1997) 3, pp. 69-74.

Kitchenham, B.; Mendes, E.: *Why comparative effort prediction studies may be invalid*. In: Proceedings of the 5th International Conference on Predictor Models in Software Engineering. Vancouver, British Columbia, Canada 2009.

Kitchenham, B.; Pfleeger, S. L.; McColl, B.; Eagan, S.: *A case study of maintenance estimation accuracy*. In: Journal of Systems and Software, 64 (2002b) 1, pp. 57-77.

Kitchenham, B. A.; Pickard, L. M.; MacDonell, S. G.; Shepperd, M. J.: *What accuracy statistics really measure*. In: IEE Proceedings Software, 148 (2001) 3, pp. 81-85.

Klein, W. M.; Kunda, Z.: *Exaggerated Self-Assessments and the Preference for Controllable Risks*. In: Organizational Behavior and Human Decision Processes, 59 (1994) 3, pp. 410-427.

Kluckhohn, C.; Murray, H. A.: *Personality formation: The determinants*. In: Personality in nature, society, and culture. Edited by C. Kluckhohn, H. A. Murray. Edition 2, New York, 1953, pp. 53-72.

Knuth, D. E.: *The Art of Computer Programming. Volume 1: Fundamental Algorithms*. 1 Edition. Addison-Wesley Series in Computer Science and Information Processing, 1968.

Koehler, D. J.; Harvey, N.: *Confidence Judgments by Actors and Observers*. In: Journal of Behavioral Decision Making, 10 (1997) 3, pp. 221-242.

Kotler, P.; Adam, S.; Brown, L.; Armstrong, G.: *Principles of Marketing*. 3. Edition. Prentice Hall, 2006.

Krueger, C. W.: *Software Reuse*. In: ACM Computing Surveys, 24 (1992) 2, pp. 131-183.

Kuhn, T. S.: *Logic of Discovery or Psychology of Research?* In: Criticism and the Growth of Knowledge. Edited by I. Lakatos, A. E. Musgrave. Cambridge University Press, London, 1970.

Kunow, A.: *Anreizsteuerung unter Berücksichtigung von Lernkurveneffekten*. Wiesbaden, 2006.

Landis, L.; McGarry, F.; Waligora, S.; Pajerski, R.; Stark, M.; Kester, R.; McDermott, T.; Miller, J.: *Manager's Handbook for Software Development (Revision 1)*. Report SEL-84-101. NASA Software Engineering Laboratory. 1990.

Laranjeira, L. A.: *Software size estimation of object-oriented systems*. In: Software Engineering, IEEE Transactions on, 16 (1990) 5, pp. 510-522.

Larman, C.; Basili, V. R.: *Iterative and Incremental Development: A Brief History*. In: Computer, 36 (2003) 6, pp. 47-56.

Laughery Jr., K. R.; Laughery Sr., K. R.: *Human factors in software engineering: a review of the literature*. In: Journal of Systems and Software, 5 (1985) 1, pp. 3-14

Lederer, A. L.; Prasad, J.: *Information systems software cost estimating: a current assessment*. In: Journal of Information Technology, 8 (1993) 1, pp. 22-33.

Lee, A. S.: *A Scientific Methodology for MIS Case Studies*. In: MIS Quarterly, 13 (1989) 1, pp. 32-50.

Lee, A. S.: *Integrating Positivist and Interpretive Approaches to Organizational Research*. In: Organization Science, 2 (1991) 4, pp. 342-365.

Lee, A. S.: *Rigor and Relevance in MIS Research: Beyond the Approach of Positivism Alone*. In: MIS Quarterly, 23 (1999) 1, pp. 29-33.

Lee, A. S.; Baskerville, R. L.: *Generalizing Generalizability in Information Systems Research*. In: Information Systems Research, 14 (2003) 3, pp. 221-243.

Lehman, M. M.: *Programs, Life Cycles and Laws of Software Evolution*. In: Proceedings of the IEEE. 1980, pp. 1060-1076.

Lehman, M. M.: *Laws of Software Evolution Revisited*. In: Proceedings of the Software Process Technology - Proceedings of the 5th European Workshop. Nancy, France 1996, pp. 108–124.

Lehman, M. M.; Ramil, J. F.: *Evolution in Software and Related Areas*. In: Proceedings of the Proceedings of the 4th International Workshop on Principles of Software Evolution. International Conference on Software Engineering. Vienna, Austria 2001, pp. 1-16.

Leong, F. T. L.; Austin, J. T.: *The Psychology Research Handbook: A Guide for Graduate Students and Research Assistants*. 2. Edition. Sage, 2005.

Levenshtein, V. I.: *Binary codes capable of correcting deletions, insertions, and reversals*. In: Soviet Physics Doklady, 10 (1966).

Liebchen, G. A.; Shepperd, M.: *Software productivity analysis of a large data set and issues of confidentiality and data quality*. In: Proceedings of the Software Metrics, 2005. 11th IEEE International Symposium. 2005, pp. 3 pp.

Linda, A.; Chester, A. I.; Nancy, Y.; Anna, A. R.: *Group Learning Curves: The Effects of Turnover and Task Complexity on Group Performance*. In: Journal of Applied Social Psychology, 25 (1995) 6, pp. 512-529.

LINDO Systems: *LINDO Systems - Optimization Software: Integer Programming, Linear Programming, Nonlinear Programming, Global Optimization*. 2009. Available at http://www.lindo.com/. Last access on 2009-07-29.

Linstone, H. A.; Turoff, M.: *Delphi Method: Techniques and Applications*. Addison-Wesley, 1975.

Linstone, H. A.; Turoff, M.: *Delphi Method: Techniques and Applications*. 2002. Available at http://www.is.njit.edu/pubs/delphibook/.

Little, T.: *Schedule estimation and uncertainty surrounding the cone of uncertainty*. In: Software, IEEE, 23 (2006) 3, pp. 48-54.

Madachy, R. J.: *A Software Project Dynamics Model for Process Cost, Schedule and Risk Assessment (Dissertation)*. University of Southern California, 1994.

Mansfield, E.; Rapoport, J.; Schnee, J.; Wagner, S.; Hamburger, M.: *Research and innovation in the modern corporation*. New York, 1971.

Martin, J.: *Rapid application development*. Macmillan Publishing Co., Inc., 1991.

Matson, J. E.; Barrett, B. E.; Mellichamp, J. M.: *Software development cost estimation using function points*. In: IEEE Transactions on Software Engineering, 20 (1994) 4, pp. 275-287.

Maxwell, K. D.; Forselius, P.: *Benchmarking software development productivity*. In: IEEE Software, 17 (2000) 1, pp. 80-88.

Maxwell, K. D.; Van Wassenhove, L.; Dutta, S.: *Software Development Productivity of European Space, Military and Industrial Applications*. In: IEEE Transactions on Software Engineering, 22 (1996) 10, pp. 706-718.

McConnell, S.: *Rapid Development: Taming Wild Software Schedules*. Microsoft Press Redmond, WA, USA, 1996.

McConnell, S.: *Code Complete*. 2. Edition. Microsoft Press, Washington, 2004.

McConnell, S.: *Software Estimation: Demystifying the Black Art*. Microsoft Press, Redmond, WA, 2006.

Metcalfe, R. M.; Boggs, D. R.: *Ethernet: distributed packet switching for local computer networks*. In: Communications of the ACM, 19 (1976) 7, pp. 395-404.

Mills, H.: *Top-down programming in large systems*. In: Debugging Techniques in Large Systems. Edited by R. Rustin. Englewood Cliffs, N.J., Prentice-Hall, 1971.

Molokken-Ostvold, K.; Jorgensen, M.: *Group Processes in Software Effort Estimation*. In: Empirical Software Engineering, 9 (2004) 4, pp. 315-334.

Molokken, K.: *Effort and schedule estimation of software development projects (PhD thesis)*. Oslo, 2004.

Molokken, K.; Jorgensen, M.: *A Review of Surveys on Software Effort Estimation*. In: Proceedings of the Proceedings of the 2003 International Symposium on Empirical Software Engineering. 2003, pp. 223-230.

Moore, G.: *Cramming more components onto integrated circuits*. In: Electronics Magazine, 38 (1965) 8.

Morin, L. H.: *Estimation of Resources for Computer Programming Projects*. University of North Carolina, Chapel Hill, NC, 1973.

Mukhopadhyay, T.; Vicinanza, S. S.; Prietula, M. J.: *Examining the Feasibility of a Case-Based Reasoning Model for Software Effort Estimation*. In: MIS Quarterly, 16 (1992) 2, pp. 155-171.

Myers, G. J.: *A controlled experiment in program testing and code walkthroughs/inspections*. In: Communications of ACM, 21 (1978) 9, pp. 760-768.

Myers, M. D.; Newman, M.: *The qualitative interview in IS research: Examining the craft*. In: Information and Organization, 17 (2007) 1, pp. 2-26.

NASA Office of Human Capital: *Competency Management System (CMS)*. 2006. Available at http://ohcm.gsfc.nasa.gov/cms/home.htm. Last access on 2008-09-27.

Nash, S. H.; Redwine, S. T.: *People and organizations in software production: a review of the literature*. In: ACM SIGCPR Computer Personnel, 11 (1988) 3, pp. 10-21.

Naur, P.; Randell, B.: *Software Engineering: Report on a conference sponsored by the NATO Science Committee*. Report. Garmisch, Germany 1969.

Nelson, E. A.: *Management Handbook for the Estimation of Computer Programming Costs*. Report AD-A648750. Systems Development Corp. 1966.

Newby-Clark, I. R.; Ross, M.; Buehler, R.; Koehler, D. J.; Griffin, D.: *People Focus on Optimistic Scenarios and Disregard Pessimistic Scenarios While Predicting Task Completion Times*. In: Journal of Experimental Psychology: Applied, 6 (2000) 3, pp. 171-181.

Niehaves, B.; Dreiling, A.; Ribbert, M.; Holten, R.: *Conceptual Modeling - An Epistemological Foundation*. In: Proceedings of the American Conference on Information Systems. New York, NY, USA 2004, pp. 4232-4242.

Norden, P. V.: *Useful tools for project management*. In: Management of Production. Edited by M. K. Starr. Baltimore, 1970, pp. 71-101.

Object Management Group: *Unified Modeling Language Specification 1.5*. 2004. Available at http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML . Last access on 2009-06-24.

Park, R. E.: *Software Size Measurement: A Framework for Counting Source Statements*. Report CMU/SEI-92-TR-20. Software Engineering Institute. Pittsburg, PA 1992.

Parkinson, C. N.: *Parkinson's Law*. In: The Economist, November (1955).

Parkinson, C. N.: *Parkinson's Law and other Studies in Administration*. Boston, 1957.

Parnas, D. L.; Clements, P. C.: *Rational design process: How and why to fake it*. In: IEEE Transactions on Software Engineering, 12 (1986) 2, pp. 251-257.

Paynter, J.: *Project estimation using screenflow engineering*. In: Proceedings of the International Conference on Software Engineering: Education and Practice. Dunedin, New Zealand 1996, pp. 150-159.

Pett, M. A.: *Nonparametric Statistics in Health Care Research: Statistics for Small Samples and Unusual Distributions*. 2. Edition. Sage, 1997.

Pham, H.: *Software Reliability*. Springer-Verlag New York, Inc., 1999.

Phan, D.; Vogel, D.; Nunamaker, J.: *The search for perfect project management*. In: Computerworld, 22 (1988) 39, pp. 95-100.

Poppendieck, M.; Poppendieck, T.: *Lean Software Development: An Agile Toolkit*. 1 Edition. Addison-Wesley Longman, Amsterdam, 2003.

Popper, K. R.: *The Logic of Scientific Discovery*. Hutchinson, London, UK, 1959.

Popper, K. R.: *The Logic of Scientific Discovery*. Unwin Hyman, London, 1980.

Premraj, R.; Shepperd, M.; Kitchenham, B.; Forselius, P.: *An Empirical Analysis of Software Productivity over Time*. In: Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS'05). 2005.

Premraj, R.; Twala, B.; Mair, C.; Forselius, P.: *Productivity of Software Projects by Business Sector: An Empirical Analysis of Trends*. In: Proceedings of the 10th IEEE International Software Metrics Symposium. Chicago, USA 2004.

Prototype Core Team: *Prototype JavaScript framework: Easy Ajax and DOM manipulation for dynamic web applications*. 2009. Available at http://www.prototypejs.org/. Last access on 2009-06-12.

Putnam, L. H.: *A General Empirical Solution to the Macro Software Sizing and Estimating Problem*. In: IEEE Transactions on Software Engineering, SE-4 (1978) 4, pp. 345-361.

Putnam, L. H.; Myers, W.: *Measures for Excellence: Reliable Software on Time, within Budget*. Prentice Hall PTR, Englewood Cliffs, N.J., 1991.

Quantitative Software Management: *QSM Function Point Programming Languages Table*. 2005. Available at http://www.qsm.com/FPGearing.html. Last access on 2009-01-06.

Quantitative Software Management: *QSM Home: SLIM Software Cost Estimation and Project Management Tools, Training, and Services*. 2008. Available at http://www.qsm.com/index.html. Last access on 2008-12-01.

Randell, B.: *Software engineering in 1968*. In: Proceedings of the 4th international conference on Software Engineering. 1979, pp. 1-10.

Randell, B.: *The 1968/69 NATO Software Engineering Reports*. 1996. Available at http://homepages.cs.ncl.ac.uk/brian.randell/NATO/NATOReports/index.html. Last access on 2008-12-16.

Recker, J. C.: *Understanding Process Modelling Grammar Continuance: A study of the Consequences of Representational Capabilities*. Queensland University of Technology. Brisbane 2008.

Reel, J. S.: *Critical success factors in software projects*. In: Software, IEEE, 16 (1999) 3, pp. 18-23.

Resig, J.: *jQuery: The Write Less, Do More, JavaScript Library*. 2009. Available at http://jquery.com/. Last access on 2009-06-12.

Roberts, E. B.: *A Simple Model of R&D Project Dynamics*. In: Managerial Applications of System Dynamics. Edited by E. B. Roberts. MIT Press, Cambridge, MA, 1981.

Rosenkranz, C.; Holten, R.: *On the Role of Conceptual Models in Information Systems Research – From Engineering to Research*. In: Proceedings of the 15th European Conference on Information Systems (ECIS 2007). St. Gallen, Switzerland 2007.

Royce, W. W.: *Managing the Development of Large Software Systems: Concepts and Techniques*. In: Proceedings of the Proceedings of WesCon. 1970.

Rubey, R. L.: *High Order Languages for Avionics Software - A Survey, Summary, and Critique*. In: Proceedings of the NAECON. 1978.

Rubin, H. A.: *Macro-estimation of software development parameters: The ESTIMACS system*. In: Proceedings of the Conference on software development tools, techniques, and alternatives (SOFTAIR). Arlington, Virginia 1983.

Rumelhart, D. E.; Hinton, G. E.; Williams, R. J.: *Learning Internal Representations by Error Propagation*. In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Edited, MIT Press, Cambridge, MA, 1986, pp. 318-362

Runeson, P.: *Using Students as Experiment Subjects - An Analysis on Graduate and Freshmen Student Data*. In: Proceedings of the Empirical Assessment in Software Engineering. 2003, pp. 95-102.

Sääksjärvi, M.; Lassila, A.; Nordström, H.: *Evaluating the Software as a Service Business Model: From CPU Time-Sharing to Online Innovation Sharing*. In: Proceedings of the IADIS International Conference e-Society 2005, Qawra, Malta (2005), pp. 177-186.

Sackman, H.; Erikson, W. J.; Grant, E. E.: *Exploratory experimental studies comparing online and offline programming performance*. In: Communications of the ACM, 11 (1968) 1, pp. 3-11.

Schach, S. R.: *Software engineering*. 2nd Edition. Aksen Associates & Irwin, Boston, MA, 1992.

Schach, S. R.: *Object-Oriented and Classical Software Engineering*. Sixth Edition. McGraw Hill, New York, 2005.

Scherer, F. M.: *Firm Size, Market Structure, Opportunity, and the Output of Patented Inventions*. In: American Economic Review, 53 (1965).

Schmidt, D. C.: *Model-Driven Engineering*. In: IEEE Computer, 39 (2006) 2, pp. 25-31.

Schmookler, J.: *The Size of Firm and the Growth of Knowledge*. In: Patents, Invention, and Economic Change. Edited by J. Schmookler. Cambridge, MA, 1972.

Schneeweiß, C.: *Einführung in die Produktionswirtschaft*. 7 Edition. Springer, Berlin, Heidelberg, New York, 1999.

Schneider, G.; Winters, J. P.: *Applying Use Cases: A Practical Guide*. Addison Wesley, Boston, MA, 1998.

Schwaber, K.: *Scrum Development Process*. In: Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications. 1995, pp. 117-134.

Schwartz, J.: *Analyzing Large-Scale System Development*. In: Proceedings of the 1968 NATO Conference. 1968.

Sekaran, U.: *Research Methods for Business: A Skill Building Approach*. 4. Edition. Wiley, 2003.

Selby, R. W.: *Software engineering: Barry W. Boehm's lifetime contributions to software development, management, and research*. Wiley & Sons, 2007.

Shepperd, M.; Schofield, C.: *Estimating Software Project Effort using Analogies*. In: IEEE Transactions on Software Engineering, 23 (1997) 11, pp. 736-743.

Shepperd, M.; Schofield, C.; Kitchenham, B.: *Effort estimation using analogy*. In: Proceedings of the Proceedings of the 18th International Conference on Software Engineering. 1996, pp. 170-178.

Simon, H. A.: *Making Management Decisions: The Role of Intuition and Emotion*. In: Academy of Management Executive, 1 (1987), pp. 57-63.

Simpson, E. H.: *The Interpretation of Interaction in Contingency Tables*. In: Journal of the Royal Statistical Society. Series B (Methodological), 13 (1951) 2, pp. 238-241.

Simpson, J.; Weiner, E.: *The Oxford English Dictionary*. Second Edition. 1989.

Sjøberg, D. I. K.; Hannay, J. E.; Hansen, O.; Kampenes, V. B.; Karahasanovic, A.; Liborg, N. K.; Rekdal, A. C.: *A survey of controlled experiments in software engineering*. In: IEEE Transactions on Software Engineering, 31 (2005) 9, pp. 733-753.

Smidts, C.; Huang, X.; Widmaier, J. C.: *Producing reliable software: an experiment*. In: Journal of Systems and Software, 61 (2002) 3, pp. 213-224.

Sneed, H.: *Die Data-Point-Methode*. In: ONLINE, ZfD, 90 (1990) 5, pp. 48.

Sneed, H. M.: *Schätzung der Entwicklungskosten von objektorientierter Software*. In: Informatik-Spektrum, 19 (1996) 3, pp. 133-140.

Software Productivity Research: *Function Points or Feature Points?* 2009a. Available at http://www.spr.com/products/choosing.shtm. Last access on 2009-01-08.

Software Productivity Research: *What Are Feature Points?* 2009b. Available at http://www.spr.com/products/feature.shtm. Last access on 2009-01-08.

Sommerville, I.: *Software Engineering*. 5th Edition. Addison-Wesley, Reading, MA, 1996.

Sommerville, I.: *Software Engineering*. 8. Edition. Addison Wesley, 2006.

Spector, A.; Gifford, D.: *A computer science perspective of bridge design*. In: Communications of the ACM, 29 (1986) 4, pp. 267-283.

St-Pierre, D.; Maya, M.; Abran, A.; Desharnais, J. M.; Bourque, P.: *Full Function Points: Counting Practices Manual*. Report 1997-04. Software Engineering Management Research Laboratory and Software Engineering Laboratory in Applied Metrics, University of Quebec. Montreal 1997.

Standish Group International: *The Chaos Report*. 1994. Available at www.standishgroup.com. Last access on.

Standish Group International: *Extreme CHAOS*. 2001. Available at http://www.standishgroup.com. Last access on.

Steinberg, S. J.; Steinberg, S. L.: *Geographic Information Systems for the Social Sciences: Investigating Space and Place*. Sage Publications, Inc., 2005.

Stensrud, E.; Myrtveit, I.: *Human Performance Estimating with Analogy and Regression Models: An Empirical Validation*. In: Proceedings of the Fifth International Symposium on Software Metrics (METRICS'98). 1998, pp. 205-213.

Sternberg, R. J.; Berg, C. A.: *Intellectual development*. Cambridge University Press, 1992.

Stroustrup, B.: *The C++ Programming Language*. 3rd Edition. Addison-Wesley, 2000.

Sun Microsystems Inc.: *Developer Resources for Java Technology*. 2008. Available at http://java.sun.com/. Last access on 2008-09-15.

Surowiecki, J.: *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations*. Little, Brown, 2004.

Symons, C. R.: *Function Point Analysis: Difficulties and Improvements*. In: IEEE Transactions on Software Engineering, 14 (1988) 1, pp. 2-11.

Takeuchi, H.; Nonaka, I.: *The new product development game*. In: Harvard Business Review, 64 (1986) 1, pp. 137-146.

Tassc: *Tassc Software Solutions On Time*. 2009. Available at http://www.tassc-solutions.com/index.htm. Last access on 2009-01-09.

Taylor & Francis Group: *Journal of Construction Management and Economics*. 2008. Available at http://www.tandf.co.uk/journals/titles/01446193.html. Last access on 2008-10-02.

The PHP Group: *PHP: Hypertext Preprocessor*. 2008. Available at http://www.php.net/. Last access on 2008-09-05.

Tichy, W. F.: *Hints for Reviewing Empirical Work in Software Engineering*. In: Empirical Software Engineering, 5 (2000) 4, pp. 309-312.

Touran, A.; Lopez, R.: *Modeling Cost Escalation in Large Infrastructure Projects*. In: Journal of Construction Engineering & Management, 132 (2006) 8, pp. 853-860.

Tukey, J. W.: *Exploratory Data Analysis*. Addison-Wesley, Reading, MA, 1977.

Tukey, J. W.: *We need both Exploratory and Confirmatory*. In: The American Statistician, 34 (1980) 1, pp. 23-25.

Turner, M.; Budgen, D.; Brereton, P.: *Turning Software into a Service*. In: Computer, 36 (2003) 10, pp. 38-44.

U.S. Department of Energy: *An Analysis of Nuclear Power Plant Operating Costs: A 1995 Update*. 1995.

USC-CSE: *COCOMO II - Model Definition Manual*. 2008. Available at http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf. Last access on 2008-12-12.

van Deursen, A.; Klint, P.; Visser, J.: *Domain-specific languages: an annotated bibliography*. ACM Press New York, NY, USA, 2000.

Verner, J.; Tate, G.: *Estimating size and effort in fourth-generation development*. In: Software, IEEE, 5 (1988) 4, pp. 15-22.

Verner, J. M.; Tate, G.: *A Model for Software Sizing*. In: Journal of Systems and Software, 7 (1987) 2, pp. 173-177.

Verner, J. M.; Tate, G.; Jackson, B.; Hayward, R. G.: *Technology dependence in function point analysis: a case study and critical review*. Proceedings of the 11th international conference on Software engineering. Pittsburgh, Pennsylvania, United States 1989.

Vogelezang, F.: *Using COSMIC-FFP for sizing, estimating and planning in an ERP environment*. In: Applied Software Measurement Proceedings of the IWSM/MetriKon 2006. Edited by A. Abran, M. Bundschuh, G. Büren, R. R. Dumke. Potsdam, 2006.

W3C: *HyperText Markup Language (HTML) Home Page*. 2004. Available at http://www.w3c.org/MarkUp/. Last access on 2004-11-11.

Wagner, C. H.: *Simpson's Paradox in Real Life*. In: The American Statistician, 36 (1982) 1, pp. 46-48.

Walsham, G.: *Interpretive case studies in IS research: nature and method*. In: European Journal of Information Systems, 4 (1995) 1, pp. 74-81.

Walston, C. E.; Felix, C. P.: *A Method of Programming Measurement and Estimation*. In: IBM Systems Journal, 16 (1977) 1, pp. 54-73.

Weber, R.: *The Rhetoric of Positivism Versus Interpretivism: A Personal View*. In: MIS Quarterly, 28 (2004) 1, pp. iii-xii.

Weinberg, S. L.; Abramowitz, S. K.: *Statistics Using SPSS: An Integrative Approach*. 2. Edition. Cambridge University Press, 2008.

Weinstein, N. D.: *Unrealistic optimism about future life events*. In: Journal of Personality and Social Psychology, 39 (1980) 5, pp. 806-820.

Weinwurm, G. F.: *On the Management of Computer Programming*. Auerbach Publications, 1970.

Whitmire, S. A.: *3D Function Points: Scientific and Real-Time Extensions to Function Points*. In: Proceedings of the Pacific Northwest Software Quality Conference. 1992.

Wilcoxon, F.: *Individual Comparisons by Ranking Methods*. In: Biometrics Bulletin, 1 (1945) 6, pp. 80-83.

Wilkes, M. V.; Wheeler, D. J.; Gill, S.: *The Preparation of Programs for an Electronic Digital Computer*. 1952.

Williams, L.; Cockburn, A.: *Agile Software Development: It's about Feedback and Change*. In: Computer, 36 (2003) 6, pp. 39-43.

Wright, T. P.: *Factors Affecting the Cost of Airplanes*. In: Journal of Aeronautical Sciences, 128 (1936) 3, pp. 122-128.

Yeaple, R. N.: *Why are small R&D organizations more productive?* In: Engineering Management, IEEE Transactions on, 39 (1992) 4, pp. 332-346.

Yelle, L. E.: *The Learning Curve: Historical Review and Comprehensive Survey*. 1979.

Yin, R. K.: *Case Study Research: Design and Methods*. 3 Edition. SAGE Publications, Thousand Oaks, CA, USA et al., 2003.

Zenger, T. R.: *Explaining Organizational Diseconomies of Scale in R&D: Agency Problems and the Allocation of Engineering Talent, Ideas, and Effort by Firm Size*. In: management Science, 40 (1994) 6, pp. 708-729.

Zmud, R. W.: *Individual Differences and MIS Success: A Review of the Empirical Literature*. In: management Science, 25 (1979) 10, pp. 966-979.

# APPENDIX

## A.1 SQL Trainer Exercises

| Original exercise number / Altered exercise number | Exercise text (German & English) / Solution |
|---|---|
| 1 / 2 | German: *Listen Sie alle Einträge über alle Spalten der Tabelle “sales” auf. Hinweis für die folgenden Aufgaben: Die Datensätze der Tabelle "sales" sind als einzelne Geschäftsvorfälle zu verstehen.*<br><br>English: *List all records of the table “sales“. Please note: The records of the table “sales” represent business transactions.* |
| | `SELECT    *`<br>`FROM      sales` |
| 2 / 1 | German: *Listen Sie alle Einträge über alle Spalten der Tabelle “sales” auf, aufsteigend sortiert nach der Verkaufsmenge “UnitsSold”.*<br><br>English: *List all records of the table “sales”, sorted by “UnitsSold” in ascending order.* |
| | `SELECT    *`<br>`FROM      sales`<br>`ORDER BY  UnitsSold` |
| 3 / 4 | German: *Listen Sie alle Einträge über alle Spalten der Tabelle “sales” auf, jedoch nur für Umsätze in der Filiale Nr.4 (“StoreNo”).*<br><br>English: *List all records, which refer to sales of store no. 4.* |
| | `SELECT    *`<br>`FROM      sales`<br>`WHERE     StoreNo = 4` |
| 4 / 9 | German: *Listen Sie alle Spalten der Tabelle “sales” auf, jedoch nur für Umsätze in der Region “North” und “South.”*<br><br>English: *List all records, which refer to sales in the regions “North” and “South.”* |
| | `SELECT    *`<br>`FROM      sales`<br>`WHERE     SalesRegion = "North" OR SalesRegion = "South"` |

| Original exercise number / Altered exercise number | Exercise text (German & English) / Solution |
|---|---|
| 5 / 5 | German: *Listen Sie alle Spalten der Tabelle “sales” auf, jedoch nur für Geschäftsvorfälle, wo die Verkaufsmenge (“UnitsSold”) zwischen einschließlich 10 und einschließlich 20 liegt.*<br><br>English: *List all records, which refer to sales with a sales volume between 10 and 20.* |
| | `SELECT     *`<br>`FROM       sales`<br>`WHERE      UnitsSold >= 10 AND UnitsSold <= 20`<br>`/* or */`<br>`WHERE      UnitsSold BETWEEN 10 AND 20` |
| 6 / 3 | German: *Listen Sie die Spalten ItemNo, ItemDescription, UnitPrice und UnitsSold der Tabelle “sales” auf, absteigend sortiert nach dem Stückpreis “UnitPrice“.*<br><br>English: *List all records of the table “sales”, sorted by “UnitPrice” in descending order. Only the columns ItemNo, ItemDescription, UnitPrice, and UnitsSold should be listed.* |
| | `SELECT     ItemNo, ItemDescription, UnitPrice, UnitsSold`<br>`FROM       sales`<br>`ORDER BY   UnitPrice DESC` |
| 7 / 7 | German*: Listen Sie die Spalten ItemNo, ItemDescription, UnitPrice und UnitsSold der Tabelle “sales” auf. Eine 5. Spalte soll die Positionssumme als Produkt von Preis und Verkaufsmenge anzeigen. Die 5. Spalte soll “Total” heißen.*<br><br>English: *List the columns ItemNo, ItemDescription, UnitPrice, and UnitsSold of the table “sales“. A fifth column should calculate the item total as the product of price and sales volume. The fifth column should be renamed to “Total.”* |
| | `SELECT     ItemNo, ItemDescription, UnitPrice, UnitsSold,`<br>`           UnitPrice * UnitsSold AS Total`<br>`FROM       sales` |
| 8 / 13 | German: *Listen Sie die Spalten ItemNo, ItemDescription, UnitPrice und UnitsSold der Tabelle "sales" auf. Eine 5. Spalte soll die Positionssumme als Produkt von Preis (UnitPrice) und Verkaufsmenge (UnitsSold) anzeigen. Die 5. Spalte soll "Total" heißen. Eine 6. Spalte (genannt "Tax") soll die in der Positionssumme enthaltene Umsatzsteuer ausweisen (UnitPrice ist Bruttopreis). Der Steuersatz ist immer 19%. Hinweis: Kommastellen werden durch einen Punkt gekennzeichnet - bspw. 0.25 – nicht 0,25!*<br><br>English: *List the columns ItemNo, ItemDescription, UnitPrice, and UnitsSold of the table “sales.“ A fifth column should calculate the item total as the product of price and sales volume. The fifth column should be renamed as “Total.” A sixth column (named “Tax”) should show the included valud-added tax (UnitPrice is a gross price). The tax rate is 19%. Note: Decimal places are marked by a point, e.g., 0.25 instead of 0,25.* |
| | `SELECT     ItemNo, ItemDescription, UnitPrice, UnitsSold,`<br>`           UnitPrice * UnitsSold AS Total,`<br>`           UnitPrice * UnitsSold / 119 * 19 AS Tax`<br>`FROM       sales` |

| Original exercise number / Altered exercise number | Exercise text (German & English) / Solution |
|---|---|
| 9 / 10 | German: *Listen Sie die Artikelnummer (ItemNo) sowie den entsprechenden Umsatz - als Produkt aus UnitPrice und UnitsSold benannt als "Turnover" - pro Artikel auf.*<br><br>English: *List the item number as well as the corresponding turnover - as the product of UnitPrice and UnitsSold renamed as "Turnover" – per item.* |
| | `SELECT     ItemNo, SUM(UnitPrice * UnitsSold) AS Turnover`<br>`FROM       sales`<br>`GROUP BY   ItemNo` |
| 10 / 12 | German: *Ermitteln Sie - basierend auf der Tabelle "sales" - die Anzahl an Geschäftsvorfällen. Das Ergebnis soll "SalesTransactions" heißen.*<br><br>English: *Determine the number of transactions in the table "sales." The result should be renamed as "SalesTransactions".* |
| | `SELECT     COUNT(*) AS "SalesTransactions"`<br>`FROM       sales` |
| 11 / 8 | German: *Listen Sie - basierend auf der Tabelle "sales" - die Region (SalesRegion) sowie die entsprechende Anzahl an Geschäftsvorfällen pro Region auf (umbenannt zu "SalesTransactions".*<br><br>English: *Based on the table "sales," list all regions and the corresponding number of transactions (renamed as "SalesTransactions").* |
| | `SELECT     SalesRegion, COUNT(*) AS "SalesTransactions"`<br>`FROM       sales`<br>`GROUP BY   SalesRegion` |
| 12 / 11 | German: *Listen Sie - basierend auf der Tabelle "sales" - die Region (SalesRegion), die Filiale (StoreNo) sowie den entsprechenden Gesamtumsatz - als Produkt aus UnitPrice und UnitsSold umbenannt zu "Turnover" - pro Region und Filiale auf. Die Abfrage soll nur Geschäftsvorfälle berücksichtigen, deren Verkaufsmenge ausschließlich größer 10 ist.*<br><br>English: *Based on the table "sales," list the region (SalesRegion), the store (StoreNo), as well as the corresponding turnover per region and store. The query should only consider transactions that have a sales volume greater than 10.* |
| | `SELECT     SalesRegion, StoreNo,`<br>`           SUM(UnitPrice * UnitsSold) AS Turnover`<br>`FROM       sales`<br>`WHERE      UnitsSold > 10`<br>`GROUP BY   SalesRegion, StoreNo` |

| Original exercise number / Altered exercise number | Exercise text (German & English) / Solution |
|---|---|
| 13 / 6 | German: *Listen Sie - basierend auf der Tabelle “items” (!) - alle Produktgruppen (ItemGroup) auf. Jede Produktgruppe soll nur einmal aufgelistet werden.* |
| | English: *Based on the table “items” (!), list all product groups. Each product group should only appear once.* |
| | `SELECT     ItemGroup` |
| | `FROM       items` |
| | `GROUP BY   ItemGroup` |
| | `/* or */` |
| | `SELECT     DISTINCT ItemGroup` |
| | `FROM       items` |
| 14 / 15 | German: *Listen Sie - zunächst basierend auf der Tabelle “sales” - die Spalten ID, ItemNo, ItemDescription, UnitPrice und UnitsSold auf. Diese Geschäftsvorfälle sind anhand der Produktnummer ItemNo mit der Tabelle “items“ zu verbinden, so dass die passende Produktgruppe ItemGroup als 6. Spalte angezeigt wird. Hinweis: SQL erlaubt keine mehrdeutigen Spalten ("ambiguous columns"). Eindeutigkeit erreicht man durch die Schreibweise "tabelle.spalte"!* |
| | English: *List the columns ID, ItemNo, ItemDescription, UnitPrice, and UnitsSold of the table “sales“. A sixth column should show the corresponding product group (ItemGroup). Therefore, the transactions must be adequately joined with the table “items”. Note: SQL does not allow ambiguous columns. To make columns unique, use the notation “table.column.“* |
| | `SELECT     ID, sales.ItemNo, ItemDescription, UnitPrice,` |
| | `           UnitsSold, ItemGroup` |
| | `FROM       sales, items` |
| | `WHERE      sales.ItemNo = items.ItemNo` |
| 15 / 14 | German: *Listen Sie - basierend auf den Tabellen “items“ und “sales“ - die Produktgruppe (ItemGroup) sowie den entsprechenden Gesamtumsatz (umbenannt zu “Turnover“ als Produkt aus UnitPrice und UnitsSold) pro Produktgruppe auf.* |
| | English: *Based on the tables “items” and “sales,” list the product group (ItemGroup) and the corresponding total turnover (renamed as “turnover”) per product group.* |
| | `SELECT     ItemGroup,` |
| | `           SUM(UnitPrice * UnitsSold) AS "Turnover"` |
| | `FROM       sales, items` |
| | `WHERE      sales.ItemNo = items.ItemNo` |
| | `GROUP BY   ItemGroup` |

## A.2 Exemplary Extreme Solution Attempt (Exercise 2)

| **Query** | | **Time** |
|---|---|---|
| select * | | 24.00 |
| select * from 'sales' | *(Close to solution)* | 40.25 |
| select * from 'sales' | | 18.50 |
| select * from 'sales' | | 1.00 |
| select * from 'table sales' | | 14.5 |
| select id, storeno, salesregion, itemno, itemdescription, unitprice, unitsold from 'table sales' | | 82.00 |
| select id, storeno, salesregion, itemno, itemdescription, unitprice, unitsold from 'sales' | | 10.75 |
| select id, storeno, salesregion, itemno, itemdescription, unitprice, unitsold from 'sales' | | 1.50 |
| select * from 'sales' | *(Returned to previous attempt)* | 162.00 |
| select * from 'sales' | | 11.50 |
| select 'sales' | | 12.75 |
| select 'sales'. id, storeno, salesregion, itemno, itemdescription, unitprice, unitsold | | 44.50 |
| select 'sales'. 'id, storeno, salesregion, itemno, itemdescription, unitprice, unitsold' | | 8.00 |
| select 'sales'. 'id', 'storeno', 'salesregion', 'itemno', 'itemdescription', 'unitprice', 'unitsold' | | 31.50 |
| select 'sales', 'id', 'storeno', 'salesregion', 'itemno', 'itemdescription', 'unitprice', 'unitsold' | | 6.00 |
| select 'id', 'storeno', 'salesregion', 'itemno', 'itemdescription', 'unitprice', 'unitsold' | | 11.00 |
| select 'id', 'storeno', 'salesregion', 'itemno', 'itemdescription', 'unitprice', 'unitsold' | | 11.00 |
| select 'sales'.'id' | | 57.25 |
| select * from 'sales' | | 41.75 |
| select 'id', 'storeno' from 'sales' | | 48.00 |
| select 'id', 'storeno' | | 29.75 |
| select * from 'sales' | | 87.00 |
| select * from 'sales'.'id', 'storeno' | | 58.00 |
| select 'table sales' | | 114.00 |
| select 'id', 'storeno' | | 126.75 |
| select * from 'sales' | | 76.50 |
| select * from ' table sales' | | 15.00 |
| select * from  'table sales' | | 12.75 |
| select * from sales | *(Correct Solution)* | 10.00 |
| | **Total time:** | **1,167.50** |

## A.3 Expert Interview Script

The following information was taken at the beginning of each interview:

***Personal data***

Name, year of birth, age, nationality, years of professional experience, academic education

***Employer***

Name, city, country, turnover per year, number of employees, industry sector, founding year, typical customers, product/core competence

***Employment***

Year of employment, number of projects, position, job description

***Project Experience***

Number of projects, customers, project languages, project durations, project volumes, man-days, team sizes, project goals, responsibilities/roles in projects

The semi-structured expert interviews were based on the following questions divided into two parts:

***Part 1: "Effort Estimation"***

1) Do you have direct or indirect contact with effort estimation approaches, for example, by being responsible for producing effort estimates or by getting timeboxes for work packages?

2) Which estimation approaches do you know by name, for example, from your academic education, your job, or from literature?

3) How do you or does your employer estimate effort?

4) Do you or does your employer systematically record and analyze estimation data?

5) In addition to effort estimation, do you or does your employer try to determine the project risk?

6) How much do actual project efforts deviate from your estimates on average?

7) Are you or is your employer satisfied with estimation accuracy?

8) If not explicitly answered before: Do projects end with effort underruns, that is, a project needed less effort than estimated?

9) [Open question] What are major problems and challenges of effort estimation in your opinion?

***Part 2: "Capability of developers"***

1) Do you think there is an observable difference between software developers in terms of performance and capability?

2) Do you or does your employer try to determine the individual capabilities of developers?

3) Are individual differences of developers taken into account during effort estimation?

4) By how many percent did your general job performance increase during your career?

5) How much is the general productivity ratio between the worst and the best developers you have worked with?

6) How much does your productivity deviate per day?

7) Are there extreme negative deviations? For example, have you ever get completely stuck while working on a solution?

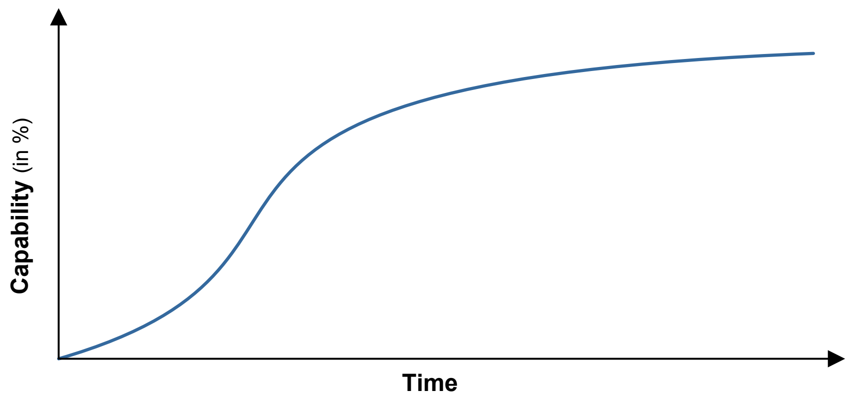
8) Is your job performance below average, on average, or above average?

9) If you receive a deadline for a work package and you recognize that you can complete that package ahead of schedule, will you adjust your working speed in order to meet the deadline on time?

10) Conversely, will you work faster if deadlines have to be met?

11) Do you agree upon the following statements: 1) Incompetent individuals will dramatically overestimate their ability and performance. 2) Incompetent individuals have a deficit in recognizing someone else's competence. 3) Incompetent individuals have a deficit in recognizing their own incompetence.

12) [With the help of the following learning curve diagram, the interviewee was asked to describe and visualize the development of his or her job-related capability over time. They started with "today" and their current capability as 100%. Then, for example, the interviewee added the time of career begin and the corresponding capability compared to today]



13) Is it possible to infinitely increase your job performance?

14) Do all developers reach the same level of capability and job performance at some point during their professional career?

15) [Open question] What are major problems and challenges of effort estimation when considering individual differences of developers? [Sub questions] How do developers affect actual project effort? How would you take individual differences of developers into account during effort estimation?

## A.4 Further Lessons Learned

The expert interviews were not solely focused on the validation of the theory. One interview question addressed general problems, challenges, and potential improvements of effort estimation (see above). For the sake of completeness, the responses of the interviewees are briefly summarized in the following table. Four interviewees named unclear and incomplete requirements as the major problem of effort estimation, since it partially invalidates effort estimates, made progress, project plans, as well as the allocation of resources.

| Interview | Response |
|---|---|
| #1 (BOB) | BOB chose unclear and incomplete requirements as the major driver for effort escalation. Moreover, he named unsteady software quality, communication problems, missing feedback loops between developers and project lead, as well as the shortage of resources. He noted that good developers are always missing due to overlapping projects, holidays, or illness. |
| #2 (TOM) | TOM also named unclear requirements. Other problems come from the unclear vision of customers as well as new technologies. Another problem is the correlation of optimism and expertise. Good developers tend to give very optimistic estimates. |
| #3 (VINCE) | VINCE asked for a validity of effort estimation. He also noted the problem of optimism. Other noted problems are: the integration of effort estimation and iterative development, missing systematic reviews of estimates, as well as the inappropriate mixing of effort, budget, and schedule. |
| #4 (MARC & PETE) | MARC named the people factor between developers and customers, unavailability of required developers, the integration of effort estimation into iterative development, missing concept fidelity, requirements changes, as well as the incorporation of switching costs. He stated that regularly switching between two parallel projects lowers productivity by 50%. PETE added that customers start to deal with projects too late. He also named the missing stability of requirements. |
| #5 (MARCUS) | MARCUS named unclear and changing requirements, the assessment of new technologies, political, and irrational decisions as the major problems of effort estimation. |
| #6 (HALE) | HALE recommended using ranges instead of single-valued estimates. He asked for an appropriate budgeting of single iterations. |
| #7 (CHRIS & KAY) | KAY named the costs of effort estimation. He noted that experience is worth more than any method. However, he also asked for a method that replaces experience. |