Adam Trendowicz · Ross Jeffery

Software Project Effort Estimation

Foundations and Best Practice Guidelines for Success

Springer

Software Project Effort Estimation

Adam Trendowicz • Ross Jeffery

Software Project Effort Estimation

Foundations and Best Practice Guidelines for Success

Springer

Adam Trendowicz
Fraunhofer Institute for
Experimental Software Engineering
Kaiserslautern
Germany

Ross Jeffery
The University of New South Wales
Sydney
New South Wales
Australia

ISBN 978-3-319-03628-1 ISBN 978-3-319-03629-8 (eBook)
DOI 10.1007/978-3-319-03629-8
Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014931020

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

Software effort estimation is one of the oldest and most important problems facing software project management; being able to plan correctly is the basis for all project management activities. One cannot manage a project without the knowledge of what resources are needed to achieve the project goals. It is an area where there has been a great deal of research in the development and fine-tuning of new models and encoding of experience in applying these models.

Today, there are a large number of models, each having different strengths and weaknesses in general and, more importantly, different strengths and weaknesses relative to the environment and context in which they are to be applied, for example, the historical data available and the kinds of factors that are relevant. At the start of a project, it is difficult to understand all the influencing factors and risks; there is a minimal amount of information available. Effort needs to be reestimated at various points in time as the project progresses. And how do you balance early effort commitment against new estimates? What trade-offs are possible?

Which models to apply under what conditions is difficult and requires a great deal of insight into the environment. As with all software engineering approaches and models, it is critical to understand the context in which the approach is to be applied, the model assumptions and context for which the model was developed (not always made clear by the model developer), and how to apply and tailor the model to your context.

This book addresses all these points and provides a large set of model types and classes, focusing on what you need to understand about your environment, what information you need to be able to apply the model, what models are most effective for a particular environment, and how you can learn from the model's application so you can evolve and improve your model over time.

The book is full of insights and useful advice on what to do and how to do it, what to be wary of, and the limitations of effort estimation. Just reading the tips contained in each chapter is a valuable experience.

The book goes beyond effort estimation and provides enormous insights into project management, in general, discussing such issues as project trade-offs, risk assessment, and organizational learning.

This is the most complete work on all aspects of software effort estimation that I have seen and provides an excellent reference for the field. It belongs on the bookshelf of every organization that needs to manage a software project. At the same time, it is an excellent text for a university course on software effort estimation, a topic that is typically insufficiently treated in most curricula.

December 2013

Victor R. Basili
University of Maryland
College Park, MD, USA

On True Success

Past successes, no matter how numerous and universal, are no guarantee of future performance in a new context.

– Henry Petroski

Failure is success if we learn from it.

– Malcolm Forbes

Success consists of going from failure to failure without loss of enthusiasm.

– Winston Churchill

To be defeated and not submit, is victory; to be victorious and rest on one's laurels, is defeat.

– Józef Piłsudski (First Marshal of Poland)

Preface

The time for action is now. It's never too late to do something.

—Antoine de Saint-Exupery.

What Is This Book About?

In this book, we focus on the estimation of software development effort. Three aspects are considered important for the proper handling of effort estimation: (1) *foundations* of software effort estimation, (2) selecting the most suitable estimation *approach*, and (3) successfully using effort estimation in specific *contexts*.

What Is This Book NOT About?

This book does not include project planning activities that typically follow effort estimation. We do not discuss such aspects as how to allocate project resources to work tasks, how to sequence work activities, how to determine critical paths, and how to resolve resource conflicts. Finally, we are not addressing project scheduling or budgeting. We refer readers interested in these subjects to books that address project management topics, for example, the PMI's (2013) Project Management Body of Knowledge (PMBOK Guide) or OGC's (2009) PRINCE2, which offer very useful overviews of common project management practices.

To Whom Is This Book Addressed?

In its very early stage, this book was intended as a collection of notes, where the most relevant estimation principles, definitions, and empirical observations, found in the literature and from experience, were gathered. In the course of time, this was shared with others. This book aims to inherit the intention of these initial notes and the needs of people they were shared with. It is addressed to those who want to take actions in order to improve their estimation practices, yet are missing (1) the necessary knowledge and understanding of estimation principles and (2) a concise reference of best practices and most common estimation approaches they can start with and adapt to their particular needs. This book assumes one prerequisite about its intended audience: it assumes that readers believe that it is never too late to do something about your estimation practices, irrespective of whatever shape they are now in.

Software Practitioners

This book is intended for all software practitioners responsible for software effort estimation and planning in their daily work. This includes primarily, but is not limited to, those who are responsible for introducing and maintaining estimation practices in a software development organization.

Students

In this book, we also appreciate the value of the old saying "as the twig is bent, so grows the tree" and address the content to students of software engineering programs, particularity project and process management courses.

How to Read This Book

We anticipated this book to be a reference guidebook you can grab whenever you need to learn or recall specific aspects of effort estimation. The way you read the book depends on your particular needs at a given moment. So before you start, think for a moment—what do you want to achieve?

- *If you want to understand the basic challenges and principles of software effort estimation*, read Chaps. 1 and 2.
- *If you want to master* the *principal concepts and techniques of existing estimation methods*, read Chaps. 3–5 and the Appendix.
- *If you want to select the most suitable estimation method for estimating software development effort in your specific context*, read Chaps. 6 and 7.
- *If additionally you want to get a quick insight into the most common estimation methods*, *including their prominent strengths and weaknesses*, read Chaps. 8–15, or only some of them if you are interested in any specific method we present there.
- *If you want to introduce a new estimation approach or improve the one you have been using*, read Chap. 16.
- *In any case, read the best-practice guidelines* we present in Chap. 17.

Moreover, each part of the book begins with a brief summary of the chapters it encompasses. Refer to these summaries to quickly decide which chapter to read.

Key Terminology Used in This Book

In this book, we use several basic terms, which in other literature and in practice are often used interchangeably. In order not to confuse the reader, we would like to start by clarifying the most important terms we will use throughout the text.

Cost Versus Effort

Although principally and intuitively different, the terms "cost" and "effort" are often used as synonyms in the software project management area. The Webster dictionary defines cost as "the amount or equivalent paid or charged for something" and effort as "conscious exertion of power" or "the total work done to achieve a particular end". In the software engineering domain, cost is defined in a monetary sense, and with respect to software development projects, it refers to partial or total monetary cost of providing (creating) certain products or services. Effort, on the other hand, refers to staff time spent on performing activities aimed at providing these products or services. In consequence, project cost includes, but is not limited to, project effort. In practice, cost includes such elements as fixed infrastructure and administrative costs for example. Moreover, dependent on the project context (e.g., currency or cost of staff unit) despite the same project effort, project cost may differ.

In the software engineering literature and practice, "cost" is often used as a synonym for "effort." One of the ways to notice the difference is to look at units used. Cost in a monetary sense is typically measured in terms of a certain currency (e.g., $, €, ¥, etc.), whereas cost in an effort sense is typically measured as staff time (e.g., person-hours, person-days, person-months, etc.).

In this book, we focus on estimating software development effort, and we consistently differentiate between cost and effort.

Estimation Versus Prediction Versus Planning

In software engineering, effort estimation, prediction, and planning are related to each other; yet, they have different meanings, that is, they refer to different project management activities. Actually, the dictionary definitions perfectly reflect the differences between these three processes:

- *Estimation*: "the act of judging tentatively or approximately the value, worth, or significance of something"
- *Prediction*: "the act of declaring or indicate in advance; especially: foretelling on the basis of observation, experience, or scientific reason"
- *Planning*: "the act or process of making or carrying out plans; *specifically*: the establishment of goals, policies, and procedures for a social or economic unit"

Estimation Versus Prediction

Both estimation and prediction contain an element of uncertainty; the first refers to approximating an actual state, whereas the latter refers to a future state. Simplifying, we may define prediction as estimating in advance. Since in software engineering, effort estimation refers to approximating development effort in advance, before development is completed, it should actually be called effort prediction. Yet, in practice, both terms are used interchangeably. In this book, we will follow this practice and use estimation and prediction as synonyms for foretelling the effort required for completing software development projects.

Prediction Versus Planning

There is, however, a significant difference between prediction and planning. Prediction refers to an unbiased, analytical process of approximating a future state. Planning, on the other hand, refers to a biased process of establishing goals with respect to the future state. Although predictions form a foundation for planning, plans do not have to be (and typically are not) the same as predictions. In the case of software development, the goal of prediction is to accurately foretell resources (such as effort) required to provide project outcomes. The goal of effort planning is, on the other hand, is to plan the project in such a way that the project goals are achieved. In other words, we plan means within a project to achieve a specific project's end.

Kaiserslautern, Germany — Adam Trendowicz
Sydney, NSW, Australia — Ross Jeffery

Acknowledgments

A number of great people and organizations have made their explicit or implicit contribution to this book by inspiring us, contributing to our knowledge, or helping us in the creation of this book. Hereby, we would like to express our gratitude to these people and organizations.

We would like to thank the reviewers of the book manuscript, Yasushi Ishigai and Mirosław Ochodek, for their valuable remarks.

Adam Trendowicz would like to express special thanks to Yasushi Ishigai for recent years of collaborative work in industrial contexts and great discussions on the industrial challenges regarding effort estimation and potential solutions to these challenges. Moreover, he would like to thank Fraunhofer Institute for Experimental Software Engineering (IESE), Kaiserslautern, Germany, for giving him an opportunity to develop their professional expertise, including software effort estimation.

Ross Jeffery would like to thank National ICT Australia (NICTA) and the University of New South Wales, who have supported his research for many years. He would also like to thank the many academic and industry colleagues who have assisted with the research in effort estimation.

We would like to convey special thanks to Mr. Ralf Gerstner and Ms. Victoria Meyer from Springer for their great support in copyediting this book.

Last but not least, we seek forgiveness from all those whose names we have failed to mention.

Disclaimer

Trademark	Subject of trademark	Trademark owner
CoBRA®	Cost Estimation, Benchmarking, and Risk Assessment	Fraunhofer Institute for Experimental Software Engineering (IESE)
GQM$^+$Strategies®		Fraunhofer Institute for Experimental Software Engineering (IESE)
CMMI®	Capability Maturity Model Integrated	Software Engineering Institute (SEI)
MS Office®	MS Word®, MS Excel®, and MS PowerPoint®	Microsoft® Corporation
PMBOK®	Project Management Body of Knowledge Guide	Project Management Institute (PMI)
PRINCE2™	Projects in Controlled Environments 2	Office of Government Commerce (OGC)

Acronyms

AC	Actual cost
ACWP	Actual cost of work performed
AHP	Analytic hierarchy process
ANGEL	Analogy estimation tool
ANN	Artificial neural networks
AVN	Analogy with virtual neighbor
BBN	Bayesian belief network
BCWP	Budgeted cost of work performed
BCWS	Budgeted cost of work scheduled
BRACE	Bootstrap-based analogy cost estimation
BRE	Balanced relative error
CART	Classification and regression trees
CASE	Computer-aided software engineering
CI	Confidence interval
CMMI	Capability maturity model integrated
CoBRA	Cost estimation, benchmarking, and risk assessment
COCOMO	Constructive cost model
COTS	Commercial off-the-shelf
DAG	Directed acyclic graph
DBMS	Database management system
EF	Experience factory
EQF	Estimating quality factor
EO	Effort overhead
ESA	European Space Agency
EV	Earned value
EVM	Earned value management
FP	Function points
FPA	Function points analysis
GAO	US Government Accountability Office
GP	Genetic programming
GQM	Goal-question-metric
IEEE	Institute of Electrical and Electronics Engineers
IFPUG	International Function Point Users Group

IRQ	Interquartile range
ISBSG	International Software Benchmarking Standards Group
JPD	Join probability distribution
KPA	Key process area
LAD	Least absolute deviation
LMS	Least median of squares
LOC	Lines of code
MCDA	Multi criteria decision analysis
MIS	Management information systems
MMRE	Mean magnitude of relative error
MRE	Magnitude of relative effort
MSE	Mean squared error
MSWR	Manual stepwise regression
NPT	Node probability table
OEM	Original equipment manufacturer
OLS	Ordinary least squares
OS	Operating system
PDCA	Plan-do-check-act
PDR	Product design review
PERT	Program evaluation and review technique
PI	Prediction interval
PMI	Project Management Institute
PMBOK	Project Management Body of Knowledge
POP	Predictive object points
PRINCE	Projects in controlled environments
PROBE	Proxy-based estimation
PV	Planned value
QA	Quality assurance
QIP	Quality improvement paradigm
QSM	Quantitative software management
RE	Relative estimation error
ROC	Rank order centroid
RR	Robust regression
SEER-SEM	Software Evaluation and Estimation of Resources-Software Estimating Model
SEI	Software Engineering Institute
SLIM	Software lifecycle management
SLOC	Source lines of code
SMART	Specific, measurable, attainable, relevant, timely
SPI	Software process improvement
SPR	Software productivity research
UCP	Use-case points
WBS	Work breakdown structure

Contents

About the Authors

Adam Trendowicz is a senior consultant at the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern, Germany, where he leads the team of "Measurement and Prediction." He received his Ph.D. in Computer Science from the University of Kaiserslautern (Germany). Dr. Trendowicz has led software cost estimation and software measurement improvement activities in software companies of different sizes and from various domains (e.g., in Germany, Japan, and India). He has been involved in functional software size estimation (Function Point Analysis) and productivity benchmarking in organizations from both industry and the public sector. Dr. Trendowicz has taught several tutorials on software cost estimation and supervised the "Software Economics and Risk Management" module within the distance master studies program "Software Engineering for Embedded Systems"—a program developed jointly by the University of Kaiserslautern and Fraunhofer IESE. Finally, Dr. Trendowicz has authored the book titled *Software Cost Estimation, Benchmarking, and Risk Assessment. The Software Decision-Makers' Guide to Predictable Software Development*. Moreover, he has coauthored more than 20 international journal and conference publications. Dr. Trendowicz's other software engineering interests include (1) project management, (2) software product quality modeling and evaluation, and (3) technology validation by means of empirical methods.

Ross Jeffery is Emeritus Professor of Software Engineering in the School of Computer Science and Engineering at the University of New South Wales and research consultant in the Systems Software Research Group in National ICT Australia (NICTA). His research interests are in the software engineering process and product modeling and improvement, electronic process guides and software knowledge management, software quality, software metrics, software technical and management reviews, and software resource modeling and estimation. His research has involved over 50 government and industry organizations over a period of 20 years and has been funded by industry, government, and universities. He has coauthored 4 books and over 190 research papers. He has served on the editorial board of the *IEEE Transactions on Software Engineering*, the *Journal of Empirical Software Engineering*, and the Wiley International Series in Information Systems. He was a founding member of the International Software Engineering Research Network (ISERN). He was elected Fellow of the Australian Computer Society for his contribution to software engineering research.

Part I
Foundations

A problem well stated is a problem half solved.

—Charles F. Kettering

In this part, we introduce the topic of software effort estimation as one of the basic elements of planning and managing software development undertakings.

Chapter 1 provides a brief introduction to software development and the application of quantitative approaches for managing software development projects. In this chapter, we also summarize typical effort estimation threats and challenges of software development projects.

Chapter 2 introduces terminological and methodological principles of software effort estimation. In this chapter, we position effort estimation within the software development environment and sketch the basic estimation process including its primary inputs and outputs.

Chapter 3 overviews common factors influencing software project effort. In this chapter, we discuss three principal groups of factors—context factors, scale factors, and effort drivers—and consider examples of the most common factors from each group. Moreover, we provide guidelines on how to reduce their negative impact on project effort in practical situations.

Chapter 4 discusses information uncertainty and estimation inaccuracy as two critical aspects of software effort estimation. In this chapter, we discuss basic types of uncertainty and common sources of uncertainty. Moreover, we provide guidelines for how to represent and handle uncertainty in effort estimation and how to reduce any negative influence of uncertainty on effort estimates. In this chapter, we particularly discuss how to handle the imperfect information upon which estimates are based. Finally, we discuss the relationship between uncertainty and change in the context of software development projects.

Chapter 5 summarizes top-down and bottom-up estimation strategies. In this chapter, we overview the strengths and weaknesses of each strategy and provide guidelines on which strategy should be used, depending on the particular estimation situation. We also discuss an estimation strategy in which multiple estimation methods are used to develop the effort estimate. Finally, we answer the question of how to aggregate multiple estimates produced by the application of either a bottom-up estimation or multiple estimation methods.

1 Challenges of Predictable Software Development

Failing to plan is planning to fail.

—Winston Churchill

Effort and cost estimation are of paramount importance for the success of software development projects. Everyday practice shows that many software organizations still propose unrealistic software costs, work within tight schedules, and finish their projects behind schedule and budget, or do not complete them at all.

In this section, we introduce software effort estimation as an essential element of a successful software development project. We look at the characteristics of software and the software engineering environment that make estimation a particularly challenging task. Finally, we try to answer the basic question of estimation, namely, "what is a good estimate?"

1.1 Software Is Getting Complex

The creation of genuinely new software has far more in common with developing a new theory of physics than it does with producing cars or watches on an assembly line.

—Terry Bollinger

Software is everywhere. Most of today's goods and services are realized, at least partially or completely with the help of software systems. Our dependency on software increases continuously. On the one hand, progress in the domains where software has traditionally been playing a key role entails increasing pressure upon software to progress. On the other hand, in domains that were traditionally reserved for hardware, software has become the major driving force of overall progress. For example, it is said that 60–90 % of advances in the automotive domain nowadays are due to software systems. Some products and services that would have traditionally been realized through "hardware" solutions are now realized through software systems. Other products and services are only possible through software systems

DOI 10.1007/978-3-319-03629-8_1, © Springer International Publishing Switzerland 2014

and could not have been realized by other means. In this way, the size and complexity of software systems in various domains has increased rapidly.

This increasing complexity of software systems entails a fundamental shift in their cost, time-to-market, functionality, and quality requirements. Software is required to support a wide variety of domains; must always be faster, more intelligent, more dependable; must require less hardware resources and be ever easier to maintain; and, and, and. The wish list is typically quite long and ends up with: "The software must cost less and come to the market before our competitors even think about something similar."

1.2 Software Development Is Getting Complex

Better, faster, cheaper. Choosing to concentrate on two of these concepts made accomplishing the third difficult or impossible.

—James E. Tomayko and Orit Hazzan

When looking at the traditional manufacturing disciplines, software practitioners may ask themselves: "If most manufacturing industries are able to control cost, schedules and quality—at least most of the time—why can't we?" One simple answer is: "because software development differs from classical manufacturing." Let us briefly go through several aspects that distinguish software development from traditional manufacturing.

Development Technologies and Paradigms Change Rapidly. Software development teams must strive to achieve software development objectives by exploiting the impressive advances in continuously changing—and thus often immature—technologies and development paradigms. In fact, mastering rapidly changing technologies and processes is often considered as the most important challenge differentiating software development from other domains. Without counting the minor changes in methods and tools, throughout the past 50 years, the software industry has roughly gone through at least four generations of programming languages and three major development paradigms.

Development Distribution Increases. Together with the increased variety of software products, technologies, and processes, development distribution is growing constantly. Development is shifting from single contractors to distributed projects, where teams are scattered across multiple companies, time zones, cultures, and continents. The global trend toward software outsourcing has led to software companies needing a reliable basis for making make-or-buy decisions or for verifying the development schedule and cost offered by contractors if they decide to buy parts of a software product.

Software Development Is Still a Largely Human-Intensive Process. Moreover, software development is a human-based activity with extreme uncertainties from the outset. Robert Glass (2002) reiterated this fact by saying: "Eighty percent of software work is intellectual. A fair amount of it is creative. Little of it is clerical." Software development depends on the capabilities of developers and on the capabilities of customers and other involved parties.

Software Products Have an Abstract Character. Probably none of the aforementioned aspects has as large an impact on the difficulty of software production as does the abstract character of software products. It is this "softness" of software products that makes software engineering differ from other, "classical," engineering domains. To create software, developers start with customer requirements and go through a sequence of transformations during which all involved parties create, share, and revise a number of abstract models of various, usually increasing, complexity. In addition, individual project tasks in a transformation sequence are usually highly interdependent. The intangible and volatile character of software products—especially requirements—makes them difficult to measure and control. This contributes to software development being a mixture of engineering, science, and art.

1.3 Project Management and Estimation Are Key Success Factors

Understanding the importance of accurate estimation, and a willingness to put in the resources . . . are vitally important to a company's success.

—Katherine Baxter

The complex and multidependent character of software development makes managing software projects a challenging task. A software project should, like any other project, be considered in terms of a business case. It should therefore lay out the reason(s) for the investment, the expected benefits of the initiative, the costs to make it happen, an analysis of the risks, and the future options that are created. A software project also requires, as one of its key success factors, effective management. It must focus on areas critical for financial success, the effective use of resources, an analysis of market potential and opportunities for innovation, the development of a learning environment, and so on.

Criteria of Project Success

The classical definition of "project success" is "a project that provides software of required functionality and quality within cost and schedule." Except for the meaning of "quality," which has been a subject of discussions for years, it is perhaps a clear definition of project success. But is it really?

In practice, success has a number of faces. Although perhaps not deemed "a success," a project that has not met some of the classical success criteria can still be far from a complete disaster. For example, if the project is canceled in a timely manner because it cannot meet the functionality and quality requirements within a reasonable budget and time, it could be classified as not having failed—under the condition that lessons learned can be applied in future projects to avoid a similar situation.

Software project management is a key project success factor, and, as aptly noted by Barry Boehm, "Poor management can increase software costs more rapidly than any other factor." A number of bad management practices may lead to failed projects, and one of the most common aspects of poor project management, which typically results in a project crisis, is poor effort estimation. Glass (2002) points to poor effort estimation as one of the two most common causes of runaway projects, besides unstable requirements. Rosencrance (2007), in her survey of more than 1,000 IT professionals, reports that two out of the three most important causes of an IT project failure are perceived to be related to poor effort estimation, in particular insufficient resource planning and unrealistic project deadlines.

Effective project management requires reliable effort and schedule estimation support. On the one hand, project managers need a reliable basis for developing realistic project effort, schedule, and cost plans. On the other hand, as project management is to a large extent a political game, they need a reliable and convincing basis for negotiating project conditions with project owners and/or customers. In the latter scenario, simple, gut-feeling estimates are definitely insufficient to justify realistic project plans against demands and expectations of other project stakeholders.

Yet, independent of these findings, many software organizations still propose unrealistic software costs, work within tight schedules, and finish their projects behind schedule and budget, or do not complete them at all.

1.4 What is a "Good Estimate"?

A good estimate is an estimate that provides a clear enough view of the project reality to allow the project leadership to make good decisions about how to control the project to hit its targets.

—Steve McConnell

The basic question of software effort estimation is "What is a good estimate?" Traditionally, effort estimation has been used for planning and tracking overall resources, such as staff required for completing a project. With this objective in mind, over the years, researchers have been pursuing an elusive target of getting 100 % accurate estimates in terms of the exact number of person-hours required to complete on a software project. Effort estimation methods that grew up on this goal focus on providing exact point estimates.

Yet, software practitioners nowadays require from effort estimation comprehensive decision support for a number of project management activities. They noticed that even the most accurate estimates are worthless if they cannot be reasonably justified to a project sponsor and customers or if they do not provide guidelines on what to do if the project is not going to meet estimates. From this perspective, one of the critical characteristics of good estimates is the additional information provided to support project decision making. Firstly, project decision makers need to identify the project areas that are responsible for increased development effort in order to have a transparent and convincing basis for renegotiating project resources and/or scope with the project sponsor. As aptly concluded by Tom Demarco (1982), the purpose of estimation "is not to solve any of the problems of actually getting a system built, but rather to make sure you encounter the fewest number of surprises as you undertake this work." Secondly, they need an indication of the effort-related development processes that can potentially be affected in order to improve productivity at low overhead—"low-hanging fruits."

Summarizing, a good estimate is one that supports the project manager to achieve successful project management and successful project completion. Thus, a good estimation method is one that provides such support without violating other project objectives such as project management overhead.

Tip

► A good estimate is one that supports project management activities such as planning and negotiation of project resources, managing changes and risks, etc. A good estimation method should thus provide—in addition to single point estimates—transparent information on project-related factors affecting development effort.

Further Reading

- R. Charette (2005), "Why software fails [software failure]," *IEEE Spectrum*, vol. 42, no. 9, pp. 42–49.

  This article looks at the status and the future of software. In the context of trends in size and complexity, it gives an overview of famous software disasters and their reasons. Among the most relevant causes of failed software projects are unrealistic or unarticulated project goals, inaccurate estimates of needed resources, and inability to handle the project's complexity.

- L.J. Osterweil (2007), "A Future for Software Engineering?," *Proceedings of the 29th International Conference on Software Engineering*, Workshop on the Future of Software Engineering, Minneapolis, MN, USA: IEEE Computer Society, pp. 1–11.

  This article identifies common trends and key challenges of software engineering practice and research. Among other items, it asks about the future of design, modeling, and quantitative quality control of something as intangible as software.

- Y. Wang (2007a), *Software Engineering Foundations: A Software Science Perspective*, CRC Software Engineering Series, vol. 2, AUERBACH/CRC Press.

  In Sect. 1.3 of his book, the author discusses general constraints of software and software engineering. He distinguishes three interrelated groups of constraints: cognitive, organizational, and resources constraints. For each group, the author lists and specifies in detail several basic constraints.

- E. Yourdon (2003), *Death March, 2nd Edition*, Prentice Hall.

  This book is one of the software engineering and project management classics, which, although not being technologically completely up-to-date now, discusses timeless traps of software project management. The author discusses reasons for software projects being what it calls "death march" projects; that is, projects that are sentenced to fail from the very beginning because of their unrealistic set up. Typical symptoms of a "death march" project are: (1) schedule, budget, and staff are about half of what would be necessary, (2) planned product scope is unrealistic, and (3) people are working 14 h a day, 6 or 7 days a week. Author suggests a number of useful solutions to avoid and, if this is not an option, to rescue death march projects.

- S. McConnell (1997), *Software Project Survival Guide, 1st Edition*, Microsoft Press.

  The book provides a set of guidelines on how to successfully perform software projects. For each major stage of software development, the author refers to the most common weaknesses that software projects typically face and discusses ways of addressing them in order to successfully get through the project.

- T. DeMarco and T. Lister (1999), *Peopleware: Productive Projects and Teams, 2nd Edition*, Dorset House Publishing Company, Inc., p. 245.

  This book discusses human aspects of software engineering. Authors show that the primary issues of software development are human, not technical.

- F. P. Brooks (1995), *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 2nd Edition*, Addison-Wesley Professional.

  This book discusses human aspects of software engineering. Fred Brooks makes a simple conjecture that an intellectual job, such as software