1

Introduction

You cannot predict nor control what you cannot measure.
—***Fenton and Pfleeger*** **[1]**

When you can measure what you are speaking about, and express it in numbers, you know something about it, but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind.
—***Lord Kelvin, 1900***

1.1 OBJECTIVE

Suppose you are a software manager responsible for building a new system. You need to tell the sales team how much effort it is going to take and how soon it can be ready. You have relatively good requirements (25 use cases). You have a highly motivated team of five young engineers. They tell you they can have it ready to ship in four months. What do you say? Do you accept their estimate or not?

Suppose you are responsible for making a go/no-go decision on releasing a different new system. You have looked at the data. It tells you that there are approximately eight defects per thousand lines of code left in the system. Should you say yea or nay?

So how did you do at answering the questions? Were you confident in your decisions?

The purpose of this textbook is to give you the tools, data, and knowledge to make these kinds of decisions. Between the two of us, we have practiced software development for over fifty years. This book contains both what we learned during those fifty years, and what we wished we had known. All too often, we were faced with situations where we could rely only on our intuition and gut feelings, rather than managing by the numbers. We hope this book will spare our readers the stress and sometimes poor outcomes that result from those types of situations.

We will provide our readers, both students and software industry colleagues, with practical techniques for the estimation and quantitative measurement of software projects. Software engineering has long been in practice both an art and a science. The challenge has been allowing for creativity while at the same time bringing strong engineering principles to bear. The software industry has not always been successful at finding the right balance between the two. We are giving you the foundation to "manage by the numbers." You can then use all of your creativity to build on that foundation.

1.2 APPROACH

This book is primarily intended to be used in a senior or graduate metrics and estimation course. It is based on a successful course in the Quantitative Software Engineering Program within the Computer Science Department at Stevens Institute of Technology. This course, which teaches measurement, metrics, and estimation, is a cornerstone of the program. Over the past few years, we have had hundreds of students, both full-time and part-time from industry, who have told us how useful it was, how they immediately were able to use it in their work/or school projects, and who helped shape the course with their feedback. One consistent feedback was the importance of exercises, problems, and projects in learning the material. We have included all of these in our text.

We believe that the projects are extremely useful: you learn by doing. Some of the projects can be quite time consuming. We found that teams of three or four students, working together, were extremely effective. Not only did students share the work load and learn from one another, but also team projects more closely simulated a real work environment, where much of the work is done in teams. For many of the projects, having the teams present their approaches to each other was a learning experience as well. As you will find, there frequently is no one right answer. Many of the projects are based on a hypothetical system for reserving theater tickets. It is introduced in an early chapter and carried throughout the text.

Although primarily intended as a textbook, we believe our colleagues in the software industry will also find this book useful. The material we have included will provide sound guidance for both establishing and evolving a software metrics program in your business. We have pulled from many sources and areas of research and boiled the information down into what we hope is an easy to read, practical reference book.

The text tackles our objectives by first providing a motivation for focusing on estimation and metrics in software engineering (Chapter 1). We then talk about how to decide what to measure (Chapter 2) and provide the reader with an overview of the fundamentals of measurement theory (Chapter 3). With that as a foundation, we identify two common areas of measurements in software: size (Chapter 4) and complexity (Chapter 5).

A key task in software engineering is the ability to estimate the effort and schedule effectively, so we also provide a foundation in estimation theory and a multitude of estimation techniques (Chapter 6).

We then introduce three additional areas of measurement: defects, reliability, and availability (Chapters 7, 8, and 9, respectively). For each area, we discuss what the area entails and the typical metrics used and provide tools and techniques for predicting and monitoring (Chapter 10) those key measures. Real-world examples are used throughout to demonstrate how theory can indeed be transformed into actual practice.

Software development is a team sport. Engineers, developers, testers, and project managers, to name just a few, all take part in the design, development, and delivery of software. The team often includes third parties from outside the primary company. This could be for hardware procurement, packaged software inclusion, or actual development of portions of the software. This last area has been growing in importance over the last decade[1] and is, therefore, deserving of a chapter (Chapter 11) on how to include these efforts in a sound software metrics program.

Knowing what and how to estimate and measure is not the end of the story. The software engineer must also be able to effectively communicate the information derived from this data to software project team members, software managers, senior business managers, and customers. This means we need to tie software-specific measures to the business' financial measures (Chapter 12), set appropriate targets for our chosen metrics through benchmarking (Chapter 13), and, finally, be able to present the metrics in an understandable and powerful manner (Chapter 14).

Throughout the book we provide examples, exercises, problems, and projects to illustrate the concepts and techniques discussed.

1.3 MOTIVATION

Why should you care about estimation and measurement in software and why would you want to study these topics in great detail?

Software today is playing an ever increasing role in our daily lives, from running our cars to ensuring safe air travel, from allowing us to complete a phone call to enabling NASA to communicate with the Mars rover, from providing us with up-to-the-minute weather reports to predicting the path of a deadly hurricane; from helping us manage our personal finances to enabling world commerce. Software is often the key component of a new product or the linchpin in a company's plans to decrease operational costs and increase profit. The ability to deliver software on time, within budget, and with the expected functionality is critical to all software customers, who either directly or indirectly are all of us.

[1]Just do an Internet search on "software outsourcing" to get a feel for the large role this plays in the software industry today. Our search came back with over 5 million hits! Better yet, mention outsourcing to a commercial software developer and have your tape recorder running.

When we look at the track record for the software industry, although it has improved over the last ten years, a disappointing picture still emerges [2].

- A full 23% of all software projects are canceled before completion.
- Of those projects completed, only 28% were delivered on time, within budget, and with all originally specified features.
- The average software project overran the budget by 45%.

Clearly, we need to change what we are doing. Over the last ten years, a great deal of work has been done to provide strong project and quality management frameworks for use in software development. Software process standards such the Capability Maturity Model® Integration developed by the Software Engineering Institute [3] have been adopted by many software providers to enable them to more predictably deliver quality software products on time and within budget. Companies are pursuing such disciplines for two reasons. First and foremost, their customers are demanding it. Customers can no longer let their success be dependent on the kind of poor performance the above statistics reflect. Businesses of all shapes and sizes are demanding proof that their software suppliers can deliver what they need when they need it. This customer demand often takes the form of an explicit requirement or competitive differentiator in supplier selection criteria. In other words, having a certified software development process is table stakes for selling software products in many markets. Second, software companies are pursuing these standards because their profitability is tied directly to their ability to meet schedule and budget commitments and drive inefficiencies out of their operations. At the heart of software process standards are clear estimation processes and a well-defined metrics program.

Even more important than being able to meet the standards, managing your software by the numbers, rather than by the seat of your pants, enables you to have repeatable results and continuous improvement. Yes, there will be less excitement and less unpaid overtime, since you will not end up as often with the "shortest schedule I can't absolutely prove I won't make." We think you can learn to live with that.

Unquestionably, software engineers need to be skilled in estimation and measurement, which means:

- Understanding the activities and risks involved in software development
- Predicting and controlling the activities
- Managing the risks
- Delivering reliably
- Managing proactively to avoid crises

Bottom line: You must be able to satisfy your customer and know what you will spend doing it.

To predict and control effectively you must be able to measure. To understand development progress, you must be able to measure. To understand and evaluate quality, you must be able to measure.

Unfortunately, measurement, particularly in software, is not always easy. How do you predict how long it will take to build a system using tools and techniques you've never used before? Just envisioning the software that will be developed to meet a set of requirements may be difficult, let alone trying to determine the building blocks and how they will be mortared together. Many characteristics of the software seem difficult to measure. How do you measure quality or robustness? How do you measure the level of complexity?

Let us look at something that seems easy to measure: time. Like software, time is abstract with nothing concrete to touch. On the surface, measuring time is quite straightforward—simply look at your watch. In actuality, this manner of measuring time is not scientifically accurate. Clock time does not take into account irregularities in the earth's orbit, which cause deviations of up to fifteen minutes, nor does it take into account Einstein's theory of relativity. Our measurement of time has evolved based on practical needs, such as British railroads using Greenwich Standard Time beginning in 1880 and the introduction of Daylight Savings Time. Simply looking at your watch, although scientifically inaccurate, is a practical way to measure time and suits our purposes quite well [4].

For software then, like time, we want measures that are practical and that we expect will evolve over time to meet the "needs of the day." To determine what these measures might be, we will first lay a foundation in measurement and estimation theory and then build on that based on the practical needs of those involved in software development.

1.4 SUMMARY

This textbook will provide you with practical techniques for the estimation and quantitative measurement of software projects. It will provide a solid foundation in measurement and estimation methods, define metrics commonly used to manage software projects, illustrate how to effectively communicate your metrics, and provide problems and projects to strengthen your understanding of the methods and techniques. Our intent is to arm you with what you will need to effectively "manage by the numbers" and better ensure the success of your software projects.

ESTIMATION AND METRICS IN THE CMMI®

The Capability Maturity Model® Integration (CMMI) is a framework for identifying the level of maturity of an organization's processes. It is the current framework supported by the Software Engineering Institute and resulted from the integration and evolution of several earlier capability maturity models. There are two approaches supported by CMMI—the continuous representation and the staged representation. Both provide a valid methodology for assessing and improving processes (see Reference 3 for details on each approach) and define levels of capability and maturity. For example, the staged approach defines five levels of organizational maturity:

1. Initial
2. Managed
3. Defined
4. Quantitatively managed
5. Optimizing

As organizations mature, they move up to higher levels of the framework. Except for Level 1, which is basically ad hoc software development, each level is made up of process areas (PAs). These PAs identify what activities must be addressed to meet the goals of that level of maturity. Software estimation and metrics indeed play a part in an organization reaching increasing levels of maturity. For example, Level 2 contains a PA called Project Planning. To fulfill this PA, the organization must develop reasonable plans based on realistic estimates for the work to be performed. The software planning process must include steps to estimate the size of the software work products and the resources needed. Another PA at Level 2 is Project Monitoring and Control. For this PA, the organization must have adequate visibility into actual progress and be able to see if this progress differs significantly from the plan so that action can be taken. In other words, there must be some way to measure progress and compare it to planned performance. At Level 4, the PAs focus on establishing a quantitative view of both the software process and the software project/product. Level 4 is all about measurement, to drive and control the process and to produce project/product consistency and quality. The goal is to use metrics to achieve a process that remains stable and predictable and to produce a product that meets the quality goals of the organization and customer. At Level 5, the focus is on continuous measurable improvement. This means that organization must set measurable goals for improvement that meet the needs of the business and track the organization's performance over time.

Clearly, a well-defined approach to estimation and measurement is essential for any software organization to move beyond the ad hoc, chaotic practices of Level 1 maturity.

REFERENCES

[1] N. Fenton and S. Pfleeger, *Software Metrics*, 2nd ed., PWS Publishing, Boston, 1997.

[2] The Standish Group, "Extreme Chaos," 2001; www.standishgroup.com/sample_ research.

[3] M. B. Chrissis, M. Konrad, and S. Shrum, *CMMI Guidance for Process Integration and Product Improvement*, SEI Series in Software Engineering, Addison-Wesley, Boston, 2003.

[4] D. Pitts, "Why is software measurement hard?" [online] 1999. Available from http://www.stickyminds.com. Accessed Jan. 6, 2005.