1

INTRODUCTION

1.1. OVERVIEW

In this chapter, we will discuss along broad, general lines what this book is about, for whom the book is meant, how the book is organized, and a reading plan. All of this is meant to get you, the reader who has come this far, sufficiently interested to carry on further. In addition, there are some reflections on why and when the narration uses “I” instead of “we” and vice versa. Lastly, I relate the reasons why I wrote this book in the first place.

1.2. WHAT IS THIS BOOK ABOUT?

This is not a book about how enterprise software should be built. This is a book about how enterprise software is built and how metrics can help make a difference. (In the next chapter, the meaning of the terms *enterprise software* and *metrics* is clarified; for the moment, let us understand them in their usual sense.) There are many books that harp on the “should” of enterprise software development. The fact that there are *many* books implies: (a) there are many strong feelings about the subject and (b) there is very little consensus. This book illustrates how simple and intuitive techniques can go a long way toward better meeting customer expectations and building systems that are easier to evolve and improve while causing developers less pain and penance.

A set of metrics and artifacts is presented which will help software engineering practitioners do their jobs better. However, no formulas are given.

In 1998, Amartya Sen won the Bank of Sweden Prize in Economic Sciences in Memory of Alfred Nobel, better known as the Nobel Prize in Economics. Sen was awarded the prize "for his contributions to welfare economics" (nobelprize.org 1998). Although Sen is a world citizen in the most eclectic sense, teaching and lecturing all over the globe, Kolkata (Calcutta), India, takes him to be very much her own. Euphoria was abounding when the news of Sen's Nobel Prize reached there. A neighbor of mine remarked with much veneration, and not a little patriotic pride, "This is such great news! But what is Professor Sen's *formula*? None of the newspapers reported it."

A formula is a nifty little thing, and formulas for happiness are in strong demand. Regrettably, formulas of such sweeping width and utility are not easy to come up with, so no such promise is made here. This book deals with mechanisms that will help you decide one way or the other as you negotiate the quagmires of building quality enterprise software. Giving you a set of heuristics or rules of thumb is very much the aim. These will complement your best judgment, and constrict your worst, but in all cases call upon your active involvement.

It is often said that a software product is never complete in the way a building or a car is. The very *softness* of software, what may be called its plasticity (refer to Section 3.2.3), ensures that software is ever caught in the quest for improvement. Thus it is no wonder the ideas in this book concerning the development of enterprise software should have room for continual refinement. Based on my own experience, I am sanguine this book's ideas will give you results. Just as I remain eager to learn from readers how they have found this book useful, I remain as receptive to know when a technique became a bottleneck or some circumstance where things did not work well. I would like readers to read this book, absorb and use the ideas, make them their own, and feel free to modify and improve them. In this era of global connectivity, a book has a new role to play. It is no longer a static repository of facts or an eternal fount of wisdom; a book now is more an instigator of interaction. In an ideal state, the interplay of ideas never had borders; the Web has just made that real. Through the Web Added Value™ downloads on the publisher's Web site (www.jrosspub.com), I plan to keep the material in this book current and remain responsive to your feedback.

In sum, this book presents a set of ideas to help practitioners build enterprise software systems that are easier to develop and better tuned to customer expectations. Many of these ideas have been applied in practice. This is meant to be a practice-oriented book which will help you understand things better and, more significantly, help you *do* things better.

The next question which begs our attention is: Who is this book meant for? Or to put it more personally: Should you be reading this book?

1.3. INTENDED AUDIENCE

Every author pines to pen a book that will become *the book* — everything for everybody. In one of Sukumar Ray's (1962) nonsense rhymes, the narrator rhapsodizes about his *notebook.* It is a compendium of every fact and all wisdom — the number of legs a grasshopper has, the diet of a cockroach, why glue feels sticky to the touch, whether jaggery goes into making soap or a firecracker, why soda has fizz but chili tastes hot, the dynamics of snoring, and what have you. The notebook's author has compiled all of these on his own, with occasional doubts cleared by pestering an elder brother. The poem ends with the readers being chided for their ignorance, which the narrator says is expected, as they have yet to read his book. Such a notebook, alas, is too big a project for our limited means. We have to be happy with subsets.

The book you have in your hands is intended for software engineers with a few years behind them in the industry — senior enough to have seen how things work and junior enough to believe things can be made to work better. Software engineers at such a level in organizations are actively involved in analysis and design and in overall charge of implementation, testing, and delivery of a software project. This role is variously named *technical lead, project leader,* or more grandly *architect.* Individuals in these roles have the independence to try out this book's ideas in their projects, along with the onus of making them work. But this does not mean you should not be reading this book if you currently do not find yourself in such a role.

If you have just entered the exciting world of enterprise software development, you will soon find yourself making decisions that will directly impact the quality and timing of customer deliverables. This book will prepare you to make those decisions with greater confidence, replicate the success when decisions go right, and when they go wrong (as they sometimes will) glean the deepest and most durable lessons. On the other hand, if you are no longer a technical lead or project leader or if the sublime tasks of an architect no longer appeal to you (in which case you have probably moved up the so-called "value-chain" and into management), this book will help you appreciate the lot of many of those with whom you need to work closely, be better equipped to help them make better software, and keep abreast of the evolving ways of software development so that some hotshot "techie" cannot take you for a ride.

I hope to interest readers at either end of a spectrum whose center makes up the core audience. For the new entrants to the industry, programming is still the entire cosmos; they sincerely believe that the success of software projects depends solely on consummate coding. This book will give them a more holistic view of software engineering. For those into the management of software projects who have been technical leads or project leaders, the book will evoke many memories. Hopefully, the dé-jà vu will foster a culture where individuals and teams can be creative with their use of metrics-based techniques.

Whenever a book is written and a publisher sought, the author faces the question whether the book can be used as a textbook in any course. I happen to feel strongly about how engineering textbooks should be written, having seen many that embody ways they should not be. A textbook in engineering is different from textbooks in other disciplines. Engineering is about doing, and engineering textbooks should ultimately talk about getting things done. It is vital to discuss the theory that goes behind the practice and equally vital to point out that practice is different from theory. To succeed, every engineer needs to learn tricks of the trade which go far beyond any textbook. This book illustrates many such tricks, which I hope will contribute to the nascent but growing collection of software engineering heuristics. Thus this book can serve as an important companion edition to software engineering course materials in any undergraduate or graduate curriculum.

I have made a conscious effort to keep jargon to a minimum; nothing puts me off more than arcane wordplay. If you do not know what a "third normal form" is, feel intimidated by the very mention of "inner classes," or are blind to the power of "pair programming," rest assured that this book will not spring such things on you without adequate reason or motivation. I only assume that the reader appreciates standard software engineering practices and the scale and complexities of enterprise software systems. Whenever subtle or involved ideas are introduced (to wit, *abstraction* in software design), I have taken care to ease things up beforehand.

If you are neither a software engineering practitioner nor a student, and do not plan on becoming either soon, even then this book might be of value to you. Software permeates our lives today as no single technology has done in the history of mankind. The story of how software is built in the large — software that handles our money, safeguards our health, lets us go safely from place to place, etc., determining our very quality of life in near transparent ways — will engage anyone with a general interest in history, society, and culture.

The books I like, irrespective of subject, have some features in common: they are written without affectations of the topic's difficulty or the author's unique gifts, they are written about circumstances the author has *lived* and not just *learned*, and they are written with simplicity and humor. These are not easy traits to emulate. In writing this book, I have been sincere in striving toward them.

1.4. ORGANIZATION OF THE BOOK

In terms of the organization of content, this book has three logical parts — Part 1: Context (Chapters 3 to 5), Part 2: Constructs (Chapters 6 to 11), and Part 3: Case Study (Chapters 12 to 16). Chapters 1 and 2 simply precede Part 1 as introductory and background material and are not numbered as a part.

Following the introductory chapter is Chapter 2, titled “Schmoozing with the Ideas.” It is kind of an up-front glossary of the main themes of the book. As I find that I seldom read glossaries, I have tried to make this chapter readable. In addition to introducing the ideas, this chapter also shows how one relates to the other in our scheme of things. Some of the words which have already been bandied about several times (metrics, enterprise software, heuristics) are clarified there in their contexts.

Part 1 serves as the overall background for the book. In Chapter 3, “Software Engineering and Metrics,” software is placed in the pantheon of engineering disciplines, the software metrics odyssey over the last few decades is discussed, and the approach of this book is summarized. Chapter 4, “Key Themes in Software Measurements,” goes over a hit parade of how software metrics should and should not be viewed or used. This is a digest of conventional wisdom, with an occasional spin from me. In Chapter 5, “Taking Your Project the Metrics Way,” the discussion focuses on some of the very real but “nontechnical” issues in taking a project to the metrics culture and making it drink. This chapter takes up such juicy issues as metrics and politics and presents a metrics charter for a project.

Part 2 presents the core ideas of this book. It covers desiderata of the major software development life cycle activities: requirements, analysis and design, implementation, and testing. For each set of activities, metrics and artifacts are introduced which help practitioners leverage the feedback mechanism of the iterative and incremental model and moderate the course of development.

Chapter 6, “Iterative and Incremental Development: A Brief Background,” goes over the salient features of this development paradigm and briefly compares it with the waterfall and agile methodologies. In Chapter 7, “Requirements: The Drivers of Software Development,” the *Correlation Matrix* and the *Mutation Score* are introduced. The former maps the artifacts of a software development project to specific requirements while the latter is a measure of how much a requirement changes over a set of iterations. Chapter 8, "Analysis and Design: On Understanding, Deciding, and Measuring," one of the longer chapters, takes a look at how metrics can help the highly interactive and reflective pursuits of analysis and design. It introduces the *Meshing Ratio* (a measure of how closely the analysis components, called *players*, link with the requirements), *AEI Score* (which reflects the extent to which a design conforms to the canons of *abstraction, elision,* and *indirection*), and *Morphing Index* (which indicates how much a design is changing between iterations). Chapter 9, "Implementation: It Takes Some Doing," is about where the rubber meets the road — when a developer gets down to writing code, line by line. It unveils the *Specific Convergence* (which measures how much of the total development effort is covered in an iteration as compared to what must finally be covered across all iterations) and the *Interface Map* (which specifies the dependencies between chunks of functionality, called *Implementation Units*, delegated to teams or individuals). Chapter 10, "Testing: Hunt for a Litmus Paper for Software," talks about the importance of testing and the gaps in conventional testing metrics. As I usually try not to be critical about something for which I cannot suggest an alternative, our take on testing differs from conventional software testing metrics: the *Exception Estimation Score* gives an idea of the likelihood the implementation of a major functional area (a *Deliverable Unit,* in our parlance) may have serious "showstopping" exceptions. The title of Chapter 11, "Putting It All Together," pretty much explains things; this chapter puts into perspective the whole of Part 2, with directions on when and how to use the metrics and artifacts.

Ideas such as those introduced in Part 2 are best explained by illustration. Part 3 is a case study that follows an enterprise software project from inception onward and shows how the techniques in Part 2 can be put into practice. The context of and characters in the case study are fictitious. To use a popular phrase, any resemblance to reality is purely accidental. But I am sure the discerning eye will find many a parallel to the "usual" circumstances of enterprise software building. (As Gerald M. Weinberg [1971] says in the preface of his *The Psychology of Computer Programming,* "…some of these experiences have had to be fictionalized to protect the innocent, and sometimes the guilty.") The chapters in Part 3 have playful names — "The Plot Thickens," "Getting into the Groove," "Diving at the Deep End," "Building Time," and "Feel of the World" — and deliberately so; because Part 3 is much storytelling, a lighter mood is not entirely out of place.

Finally, the epilogue gets into some grand rumination in the spirit of a coda and places the book in its context.

1.5. READING PLAN

Executive summaries are very hot in the software industry. You may have reams of documents detailing many subtleties, but until there is a “one-pager” that digests the whole, you are not considered done. I believe, rather naively, that the phrase “executive summary” has its roots in the kinds of summaries executives like, given their constraints on time and attention. Indeed, having the right thing summarized for the right executive can get you far. Unfortunately, however, not everything can be executively summarized or summarily executed. The author R. K. Narayan is believed to have once riposted to a reporter’s request for a short summary of his latest novel, saying there is no way he could shorten something he has taken eighty thousand words to say in the first place. The best reading plan an author can offer for his or her book is to read the whole book, from start to finish.

Yet as I often find in technical nonfiction, authors prescribe that readers go over Chapter X and then “feel free” to skip a few chapters and rejoin the trail in perfect harmony in Chapter Y. In keeping with that fashion, I offer the following advice to readers who want to avoid committing to read the whole book on the first go.

To get acquainted with the ideas with which this book will be mainly concerned, you need to read the next chapter, “Schmoozing with the Ideas.” If you want to know about the general state-of-the-art of software metrics, and how this book says something new or looks at something old in a new way, Chapter 3, “Software Engineering and Metrics,” is your destination. That is the first chapter in Part 1. The two other chapters in Part 1 build the context for Part 2. To verify whatever there is new in this book, you need to plunge into Part 2. The chapters in Part 2 are the most closely knit, and I would certainly encourage reading them in sequence. (For example, you cannot implement before you design, or at least should not, so reading Chapter 9, “Implementation: It Takes Some Doing,” before Chapter 8, “Analysis and Design: On Understanding, Deciding, and Measuring,” may not give you the best results.) If Part 2 fires up your imagination, which I am sure it will, but leaves you slightly skeptical, which I surmise it may, proceed to Part 3. Everything that is prescribed in Part 2 is illustrated by example in Part 3. In addition, Part 3 is also a tongue-in-cheek take on the ecosystem of an enterprise software development project. As in all interesting situations, understanding the ecosystem is often the foremost step in understanding the system. I have deliberately tried to make Part 3 go beyond the mere play of the metrics. Many practitioners will find many circumstances in Part 3 familiar, and this should breed an interest in the destiny of the case study project presented there. Adventurous readers may try reading Part 3 first and as their interest is piqued rewind to Parts 1 and 2, but the usual risks of adventure come with taking this route.

I have tried to make the cross-references as precise and the index as detailed as possible to help readers swing back and forth without losing their way. For those who will use the ideas in this book in their work, rereading parts of it will be helpful after an initial browse.

What I said above should qualify as a reading plan, but it does not offer you a way to absorb everything in the book in an hour and a half. This book is not like another whose title I will not mention which promised in the preface to be able to be read in its entirety during a flight from Washington, D.C. to New York.

Every reader makes his or her own reading plan to make the most out of a book. A book, in that sense, is the most customizable commodity.

1.6. OF I AND WE, AND OTHER SENSITIVITIES

Neil Armstrong's famous remark about a small step for *man* might have been cited for gender insensitivity, had the times been less lax or the occasion less exhilarating. Throughout this book I use the neuter plural "they" in talking about issues far more mundane than what Armstrong had to say. The protagonist in the case study in Part 3 is Tina, a representative of the very many young women working with such distinction in the global software industry. Many men are doing that too. I just made a random choice.

A thornier issue with the narrative has been "I" versus "we." One of the first term papers I wrote for a class had a liberal peppering of the first-person singular because I thought term papers were graded on originality. The professor called me into his office and said I should use "we." "But there is only I. No one else is in this," I interjected. "Still, we must use 'we.' 'We' sounds more academic." I smarted at the dilution of my identity at that time, but have since come to note the subtle difference in mood that "I" and "we" impart. It is usual for academic writing to assume an air of speaking *on behalf* of a putative brotherhood of believers (even if it is a gang of one), so "we" is the custom. "I" definitely has an egotistic ring to it; too much of it jars the sensibilities and makes one a spectacle, an "I specialist" that is. In this book, I do not want to sound academic or become a spectacle. As a middle ground, whenever I say something for which I solely wish to assume responsibility, I say "I." In all other cases, including when I am talking about opinions or experiences that are widely held in the software engineering community, I say "we." And when I want to be politically correct, I use neither, sending the good old passive voice to the rescue.

1.7. WHY I WROTE THIS BOOK

This chapter ends with reflections on why and how this book came to be. Writing a book is a major undertaking, especially as most books are written part-time. Moonlighting sounds very romantic, reminiscent perhaps of the *Moonlight* Sonata, but it needs much motivation. (Was Beethoven moonlighting when he came up with the eponymous sonata? Probably not.) Being too honest with the motivations may end up seeming too crass, so allow me a little discretion.

A book usually has a mission; the author may be clever or coy enough not to be forthright with it. Enlightening the readership surely qualifies as one mission, but often there are more covert ones. I first started thinking about this book almost two years ago. From that first flicker of thought to the time the book comes out as a product, there has been and will be many modifications, morphings, and modulations. And this is all in the game. All of this change allows some ideas to crystallize and others to vaporize, and finally leaves one with a core set of notions and their interconnections to explore in the book. For me, writing is very much a process of learning (I find William Zinsser's [1993] thesis in *Writing to Learn* very apt and very neglected in curricula across the world), and this book helped me see many of the things I have written about in a much clearer light.

I have written mostly about situations in enterprise software development which I experienced. This seems a very natural way to write, but there are certain difficulties. To write from experience, one has to first decide how to handle the baggage that comes with every experience. The way the past has dealt with us often colors the way we deal with the future, and this comes in the way of an objective point of view. I give many anecdotes in this book, from my own experience or hearsay from sources I trust, but I try to avoid making conclusions or, worse, judgments based on them. My experiences with building enterprise software from the bottom up in the trenches are by no means unique, and the familiarity other practitioners will recognize while reading the book should interest them more in it.

The very fact that my experiences spurred me to write a book shows there was something about them which I thought could or should have been different. Software engineering is young, in fact infant, by the standards of the other engineering disciplines (structural, mechanical, and even electrical). We do not yet have anything near the common body of knowledge which these disciplines have accumulated. Yet software engineers find themselves building critical and complex systems which influence our lives in diverse ways and even stand to affect life-and-death situations in the straightforward meaning of the term. The only way we can scale our skills to the level of preparation necessary to address such criticality and complexity is by honing a set of heuristics. Heuristics do not appear in dreams; they have to be consciously harvested from the relentless cycles of trials and errors, failures and successes that every engineer needs to go through. To do this, software engineers need to have awareness, discipline, and, curiously, a sense of history, but I had not seen these elements in many software engineers and mostly in myself. This book seeks to recoup some of that loss.

All of my experiences with building software systems and teaching and reading about them have helped me write this book. I have tested many of the techniques I describe on real-life software development projects. The results have convinced me that every metric or software development artifact has a scope of continual improvement. To improve, we must start from somewhere. Writing this book also helped me unearth many such starting points.

Lastly, through this book I hope to start a process of dialogue (and perhaps some diatribe, too!) with like- and unlike-minded people who think, talk, or do enterprise software development. The contact will certainly enrich me and may also end up helping others and furthering the discipline.

1.8. SUMMARY

This chapter is much a potpourri. We went over such meta information as what the book is about, the intended audience, organization of the chapters, a reading plan, some of the quirks of the narrative, and the author’s reasons for writing this book. If you are interested in enterprise software development by passion or profession, and want to know how simple, intuitive metrics can help make software easier to build and more useful to use, you have hit upon the right book. Read on.

In the next chapter, readers are familiarized with the main ideas in this book and their interconnections.

REFERENCES

nobelprize.org (1998). The Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel 1998. http://nobelprize.org/nobel_prizes/economics/laureates/1998.

Ray, S. (1962). *Abol Tabol,* 9th ed. Signet Press.
Weinberg, G. M. (1971). *The Psychology of Computer Programming.* Van Nostrand Reinhold.
Zinsser, W. K. (1993). *Writing to Learn.* HarperCollins.