The Software Development Edge
Essays on Managing Successful Projects
By Joe Marasco
(A book review by R. Max Wideman)

Introduction

In writing the observations in this book review, I must first confess a certain degree of conflict of interest that might affect my objectivity. Joe Marasco, the author of this book, The Software Development Edge, is a special friend of mine. "Special" because he is one of a growing group of people that is becoming increasingly common in this modern age of the Internet. I have conversed with him at length, worked with him and exchanged ideas, yet we have never actually met face-to face. Of course we both know what each other looks like, but that is as far as it goes. Such are the wonders of modern technology yet, in a way, strictly limited to what can be conveyed through the typed word.

But I digress. I find it an honor and privilege to be counted amongst Joe's acquaintances because Joe is an author of considerable intellectual prowess. He makes you think! You might believe that this book is about software development, or about project management, or about being successful – in a way it is, all of these. But the book is also about a variety of challenges that Joe has found intriguing and to which he offers solutions, the more mathematical the better. After taking a bachelor's degree in chemical engineering he earned his Ph.D. in physics from the University of Geneva. This latter no doubt laid the foundations for his excursions into mathematical modeling that has stood him well in his career in software development.

Joe Marasco is a retired senior vice-president and business-unit manager for Rational Software, a company offering software development processes and tools, and subsequently acquired by IBM. In 1998 he served as Senior VP Operations, retiring in 2003. One of Joe's greatest challenges and achievements was to take on, in 1991, a new project team to plan and deliver "Rational II" in two years and on two UNIX platforms: IBM and Sun. After seven months, a limited-function subset prototype was up and running. After sixteen months, the development team was "self-hosted" which means that the team was able to complete the development of the product using the partially completed product itself. The final product was delivered in exactly two years as promised.[1] It just goes to show – it can be done!

According to reports, one reason why Joe was so successful was because he spent a lot of time with his developers understanding the details of the products and the development problems. But he also spent a lot of time with Rational's customers, developing a keen understanding of their needs. As every product delivery is the result of many compromises, Joe was able to make well-informed judgments when it came to decision time.

In more recent years, Joe contributed articles to Rational's electronic magazine, *The Rational Edge*. These articles had a special flavor, being about Joe's practical experiences, his understanding of human responses in a technically challenging environment, and his fascination with problem solving by modeling. In a way, this book of his is a compendium of many of these articles. Some you will find refreshingly obvious, others you will find take you into the realms of serious mathematics.

For those like me, who no longer have the necessary mathematical horsepower, these sections can be skipped with out damage. As Joe himself observes, the book's chapters have several different styles. Some are expository, some are fairly analytical, and some are folksy "Socratic dialogs" between the

author and a fictional character known as Roscoe Leroy. This is Joe's device designed to get his message across. But you'll have to read the book to get to know Roscoe.

Book Structure

Joe Marasco's book is divided into six parts of four chapters each. As Joe explains in his Preface:

> **"1. General Management:** These chapters deal with topics that are useful to managers in general, and also expose the reader to my background and biases. I include them so that we have a common baseline for what follows.
> 2. **Software Differences:** In this section we have a look at those things that distinguish software development from other management challenges.
> 3. **The Project Management View:** I take the perspective that a software development project is a variant of the generic project, and, as such, amenable to classical project management techniques. On the other hand, I strive to point out what is different about software development.
> 4. **The Human Element:** I turn around in this section and look at software development from the perspective of the people who do it. Once again, I try to compare and contrast that which is similar to that which is different for software development projects.
> 5. **Thinking Laterally:** Software people come at problems from many different points of view. In this section, I expose the reader to some of the more speculative and original ideas that he or she may not have seen before.
> 6. **Advanced Topics:** The successful software development manager is like a really good pinball player: His reward for high scoring is given in free games. This additional 'stick time' leads to his becoming even more proficient. In this section I talk about some of the challenges that come with success."[2]

Chapter details are as follows:

Part 1. General Management

1. Beginning at the Beginning
2. Computational Roots
3. Mountaineering
4. Managing

Part 2. Software Differences

5. The Most Important Thing
6. Modeling
7. Coding
8. Getting It Out the Door

Part 3. The Project Management View

9. Trade-Offs
10. Estimating
11. Scheduling
12. Rhythm

Part 4. The Human Element

13. Politics
14. Negotiating
15. Signing Up
16. Compensation

Part 5. Thinking Laterally

17. History Lesson
18. Bad Analogies
19. The Refresh Problem
20. Not So Random Numbers

Part 6. Advanced Topics

21. Crisis
22. Growth
23. Culture
24. Putting It All Together

What we liked – in Parts 1 & 2

We suspect that the chapter titles listed in the previous section hardly do justice to the content. For example, in his introduction to Part 1: General Management, Joe observes that this first section of the book deals with generally useful stuff. And this Part starts with 'Beginning at the Beginning' (Chapter 1). This is not so mundane as it sounds. Not just because many people prefer to start somewhere in the middle and work both ways, nor even start at the end and work backwards, but because it introduces the subject of software development and Joe's particular view of managing it.[3] Chapter 1 also describes his "problem-solving clock" - an iterative creative cycle, as shown in Figure 1.

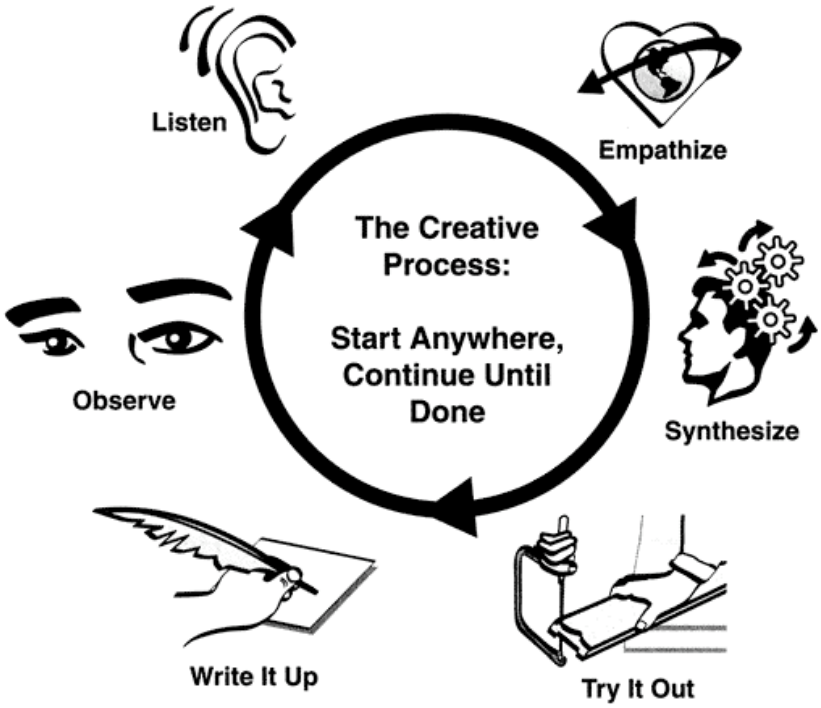

Figure 1: The iterative problem-solving clock[4]

How do you know when you are done? According to Joe: "A good guideline is to stop when there is very little significant new information obtained during the observing and listening phases."[5]

Chapter 2, Computational Roots, is an interesting foray into how computer science got started, how calculations were done pre and post the '60s and '70s. It will particularly delight older readers who remember the old slide rule with great fondness. For those who have never seen a slide rule, it is a hand-

held device for adding logarithmic numbers thus enabling multiplication and division. It had the distinct merit that you had to have a feel for the answer before you started. Blind adherence to the number expressed to the n^{th} decimal place on the computer screen, regardless of whether such accuracy had any legitimacy, was still a thing of the future.

Chapter 3 uses mountaineering as an interesting analogy for planning a software project, while the chapters in Part 2 are all about the technical aspects of software coding. However, this is not as dry as it might sound, because Joe introduces us to his imaginary "old war buddy" of his Dad's named Roscoe Leroy. Joe says of him:

> "Roscoe is one of those old salty dogs who have an opinion on just about everything."

And

> "His wisdom is simple and easy to understand. It comes in the form of very prescriptive advice: As long as you don't probe for deep theory, you'll be fine. Going with Roscoe is a leap of faith, one that benefits us all when the chips are down. With his 'been there, done that' swagger, he boasts of having finished the eighth grade, but I have it on good authority that he graduated from an excellent high school some time in the thirties. My dad explained this discrepancy to me simply by saying that Roscoe felt it unnecessary to flaunt his education."[6]

As an example, consider this exchange between Roscoe and Joe:[7]

> "Sometimes I hear software development project managers say 'We'll tackle that risk later; that'll give us much more time to think about it while we're doing all the easy stuff.' Whenever I hear that, my blood pressure goes through the roof!"

To which Joe replies:

> "Ah, Roscoe, I know where that wrong thinking comes from. It's what we tell American students about how to take tests in a time-constrained setting. Over and over again, we admonish them to do the easy stuff first, to 'get money in the bank' and reserve the remaining time at the end to work on the harder stuff. The logic is to get as much credit as you can for what you know, and 'don't leave money on the table' because you run out of time . . . [But in reality] the moral of the story is: Prioritize your risks, and attack the biggest ones first."

And Roscoe rejoins:

> "Now you're striking oil, Sonny . . . If I were behind schedule, I would feel much more comfortable going to my boss and asking for more time if I could demonstrate that my team was executing on a plan that had already squeezed out most of the risks. On the other hand, if I had to admit that not only were we behind schedule, but also that there were still high-risk items on the table, I'd be in a very weak negotiating position. I probably wouldn't get my extension, and the project might be cancelled. And maybe it should be."

What we liked – in Parts 3 & 4

Part 3 of Joe Marasco's book tackles software development from a project management view. Joe first explains the mechanics of project management using the simple notions of areas and volumes. Nevertheless, he still manages to end up with a lognormal distribution curve to represent the probability of a successful outcome. But then Joe adds a caveat:

> "To some extent, I have ignored the most important factor in any software development project: the talent of the people involved. Over and over again, I have seen that it is not

> the sheer number of people on a team that matters, but rather their skills, experience, aptitude, and character."[8]

Amen to that!

Of the infamous Standish Report on the poor success rates of IT projects, Joe has this to say:[9]

> "What about iterative development? Unfortunately, this treatment looks at the project as a 'one shot', which goes against everything we believe in with respect to iterative development. But perhaps the unusually high failure rate documented by Standish is caused by a lack of iterative development. That is, by starting with an unrealistic plan and rigidly adhering to it throughout the project, despite new data indicating we should do otherwise, we bring about our own failures."

Well said. In other words, the Standish Report focuses on the wrong metrics!

Roscoe Leroy reappears in chapter 10 on the subject of estimating. Roscoe, apparently fascinated by the concept of square roots, has this "theory" that most projects are set by some deadline and once you know that you can determine the appropriate number and duration of iterations in a software development. This theory is displayed in Figure 2.[10] Armed with this framework, it becomes possible to make a reasonable first cut estimate of the effort involved.

Duration	Roscoe's Duration	Iteration length	Number of Iterations
Two years	100 weeks	10	10
One year	49 weeks	7	7
Nine months	36 weeks	6	6
Six months	25 weeks	5	5
Four months	16 weeks	4	4
Two months	9 weeks	3	3
One month	4 weeks	2	2

Figure 2: Roscoe's Iteration Estimating Guideline

Apparently, but not surprisingly, Joe's own experience tends to validate this recommendation. However, for obvious reasons, Roscoe doesn't like projects of three-month duration![11] In chapter 11, Roscoe once again applies his square root magic to the idea of relating the quality of people's estimates to how late they are going to be in delivering by constantly calibrating their predictability. That's assuming, of course, that you can get them to estimate how long it will take for them to finish their piece of work in the first place.

Part 4 of Joe's book tackles the human side of software development projects. Chapter 13 is an excellent exposition, and discussion of principles, on politics in the project and technical environment. Joe accepts politics as a human activity like any other and defines the political process as one in which two or more people adopt a single course of action.[12] Since this is obviously essential in project work, it follows that politics is a key ingredient of project leadership. Joe goes on to describe three types of politics: "Good", "Neutral", and "Bad" and the perception of each according to three different types of people.

True to form we took exception to Joe's list of "Neutral Politics" because we happen to be that kind of person.

Part 5 provides the reader with some respite by taking some flights of fancy into the realms of history,

pseudoscience, physics and mathematics. But you'll have to buy the book to get the low down on all of that!

Downside

Chapters 14 and 15 cover "Negotiating" and "Signing up" respectively. Signing up means commitment to delivery and hence involves integrity in preparing estimates. Joe admits this is one of the most difficult areas in managing projects and, indeed, we found some positions a little hard to take. Of this, Joe observes:[13]

> "The contents of this chapter have provoked more heated discussion than the rest of the book combined. Software developers and their managers are very uncomfortable with the hard line I take on commitments. They keep trying to explain to me that 'software development is different.' "

Joe then says:

> "Well, sorry, folks, but I just don't believe it. Every area of human endeavor has risk and uncertainty. Every project ever undertaken has some new or novel element in it. All projects have schedules that are too short and resources that are inadequate. Just ask your brethren in other domains. You are not that special."
>
> "It's not that I'm unsympathetic, you understand. Software does have some peculiar problems. The most vicious of these is management's misconception that large changes can be made at the last minute through the magic of changing just one line of code. This is not reality, but fantasy. More likely, when a problem is found, it will take major rework, just like in any other area where architecture governs the result."

This is all very true. Yet like research and development projects, software development ***is*** different. The results of the effort are, for the most part, intellectual rather than tangible. That leads to two major differences:

1. Tangible products are more easily visualized, making the product easier to plan and execute
2. Tangible products, as for example in construction, generally require a trade skill rather than an intellectual skill and such people respond to a different type of management.

In fact, Joe acknowledges as much in Part 5 of his book because he introduces it by observing:[14]

> "The theme of this section is that software people think differently. Some of their ideas are speculative, maybe a little off the beaten track. It's helpful to gain the insight of seeing some of the various ways in which they look at problems."

All of which explains, albeit very briefly, why project managers "brought up" in one domain frequently have difficulty in switching and accommodating to the other.

Joe discusses appropriate pay for software professionals in Chapter 16 and, as you might have guessed, presents an interesting model of compensation. This model relates job difficulty with skill level.[15] However, it postulates a range somewhere between an individual's anxiety due to the difficulty being beyond his or her skill level on the one hand and boredom due to the job being too easy on the other. Somewhere in the middle is an area of satisfaction where there is a feeling of wellbeing that results in a high level of achievement.

Joe goes on to expand on this model and how it might be applied to software development work. In reading this section we were left with a number of unanswered issues. For example:

- The concept assumes that the organization is big enough to have a whole bucket-full of jobs of all kinds covering a range of difficulty from which to choose from and that can be dispensed to the working people to fill their working day regardless of the sequence and needs of project activities
- It also assumes that compensation is in the hands of the technology leaders who understand degree-of-job-difficulty, and not in the hands of professional "human resources" managers who know little about the challenges of software development
- And, when it comes down to actual dollar compensation figures, how do you compare the value of different work anyway?

Summary

We strongly believe that there is a distinct difference between the knowledge required to manage a project and the knowledge required to manage the technology vested in the project. The former is much more universal or "generic", but the latter is certainly not. The latter even varies from industry to industry, indeed from company to company. How else could there be market competition?

So, this book is one of the few we've seen that is for software development professionals charged with managing software development projects. It provides valuable insights into what project managers need to know about the technology in order to effectively manage such projects. Joe's last piece of good advice is worth repeating here – what to do when you are looking for a job and opportunity knocks:

> "The last item on my agenda is advice about what to do when you are looking for a job. I believe, strongly, that the biggest single factor contributing to your happiness and success in any company is how comfortable you feel with its culture and values. Almost every other variable in the equation can and will change over time: your role, your responsibilities, your direct supervisor, your organization, and your compensation. Problems in any of these areas can be addressed and fixed over time."
>
> "But if there is a fundamental incompatibility between the existing culture and your idea of what constitutes a good culture and healthy values, you will be working against something that will nag at you every day in good times and totally sink you in bad times. Remember that cultures and values change very, very, slowly. The odds are better that you will gradually adapt to the culture than that the culture will change in a direction to your liking. So unless you really enjoy swimming against the tide, look for compatibility."
>
> "How do you discover a company's true culture and values? The best way is to talk with current employees and recent ex-employees. Ask them to speak candidly about what they like and don't like. And during your formal interviewing process with the company, do two things. First, calibrate how important culture and values are by seeing whether your interviewers ask questions to determine the fit between you and their organization. If they never ask you one question in this area, beware. It means that either they are sloppy in their recruiting, or the corporate culture is extremely weak."
>
> "Second, when it is your turn to ask questions, spend as much time as you can getting

them to talk about culture and values. Don't be afraid to put them on the spot; for example, ask them flat-out what is the single most important value in the company, or what is the defining attribute of the most successful employee in the company. If they have the right culture and values, they will understand why you are asking, and will interpret your efforts as 'serious buying' questions. If you get a consistent, coherent story from almost all the people you talk to, there is reason to believe that the culture is strong: The 'code' is visible to all and understood in the same way by most. Once you have established that there is a strong culture, seeing whether your own values align with it should be somewhat easier."

Good advice indeed.

R. Max Wideman
Fellow, PMI

---

[1] Marasco, J., *The Software Development Edge*, Addison-Wesley, NY, 2005, p xx
[2] Ibid, p xxii
[3] Ibid, p1
[4] Ibid, p7
[5] Ibid, p9
[6] Ibid, pp44-45
[7] Ibid, pp52-53
[8] Ibid, p103
[9] Ibid.
[10] Ibid, p115
[11] Because of the difficulty in figuring out the square root
[12] [12] Marasco, J., *The Software Development Edge*, Addison-Wesley, NY, 2005, p147
[13] Ibid, p177
[14] Ibid, p195
[15] Ibid, p184