Foreword

Risks are everywhere in life and most assuredly during the life of software projects. Risk is not always avoidable, but it is controllable. In his book *Reducing Risk with Software Process Improvement,* Louis A. Poulin addresses the dominant risk types still seen in too many software projects. He offers practical guidance on how these risks can be managed without waxing philosophical as some writers do when addressing the need for good software processes.

One can argue that taking a risk is part of doing business. Indeed risk taking is a business choice, one that cannot and should not be stopped. But there is a major difference with intentionally taking risks, assuming risks must always be taken, or not knowing when risks are being taken.

When a business must make a constrained choice, it may take a risk to try to achieve a desirable objective. I would add that when any risk is intentionally taken, the risk taker must be willing to live with any negative outcome if the risk should manifest as a problem. If the risk taker is not willing to live with the problem should it manifest, the risk should not be taken, it should be controlled. These types of risks I would categorize as chosen, understood, and appreciated when taken. The possibility of a downside is included in the taken risk. These risks are not avoided, but are managed to minimize the problem potential.

Not all risks should be taken. Too often a risk is taken not knowing that there is a simple, practical way to avoid the risk. The worst possible risks are those not understood, seen, or appreciated by the risk taker. These risks are not managed. The risk taker does not even know they are taking a risk. The downside is not anticipated as a possibility, and when it manifests, the risk taker says, "Well, that's the way it is." Well, it is not!

I was asked to be an expert witness in a software project suit. The situation was that Company A had contracted with Company B to deliver

a software solution. However, the solution was late, overran budget, did not address all the contracted requirements Company A wanted, and was highly defective. I asked the lawyer what the defense of Company B, the contractor, was. He said they were pleading that this is the nature of software contracts; i.e., that they are late, overrun budget, misinterpret requirements, may not deliver them all, and are defective. "That's the way it is!"

I told the lawyer to defend by asking the judge if he would buy a car under these circumstances or would accept this as the state of automobile manufacturing. Clearly, he would not. But there was a time when automobiles produced by manufacturers did not work as well as the customers expected, as a result, there were many problems that were endured because there was no better solution without increasing the expense of the purchase to the customers. This had to change and did change as society became more dependent on the automobile and demanded better and safer products from the manufacturers.

Today in software, the state of poor and unacceptable quality is still too often a problem. We have improved. We have processes that when implemented lead to desirable solution delivery for both the customer and the provider. We know how to deliver good quality solutions, on time, on budget, and which address the expected customer requirements. But it is not always so. Why? Mostly because the software provider does not know or does not appreciate that it can be done and takes unnecessary risks. Software can be produced at lower cost, leading to a more competitive position for the provider, and to higher customer satisfaction which in turn leads to repeat business.

The problem is more pronounced when we understand that some software is life critical. When not life critical, it directly relates to the quality of life we all expect now that software is bundled in almost every hardware device we purchase. We as customers need and expect good, reliable software solutions.

Poulin briefly explains this industry problem and leaves the reader hoping there is a better way. There is.

While the book, "has been written to appeal to many categories of professionals," says Poulin, I believe it will be of most use to managers of software projects. A project team is needed to build and deliver the right solution, but the manager is needed to ensure the team builds the right solution. If the managers don't know how to build the right solution, believe it is possible, or understand why it is important, the team will be handicapped in doing its best.

It is sad to learn how wide poor practice remains in our industry. Poulin's analysis on poor or deficient practices is comprehensive for low maturity organizations.

Poulin lists the common poor practices as found for over 40 assessments of software projects. His data is consistent with what others have been finding for the last 40 years, which is why models like the Capability Maturity Model (CMM)* from the Software Engineering Institute were developed to help software organizations. These models are now widely used, thankfully. Unfortunately, not all organizations or managers use the models for whatever reason they find justifiable. If they have an aversion to models, then they should start with this book, which provides practical advice on critical practices.

It is a startling indictment of managers of software projects when Poulin notes that, "A deficient practice, in order to be listed, must have been observed:

> Either in at least 50 percent of the assessed organizations,
> Or on average, once per organization, by counting the number of times it was found to be the source of difficulties and dividing this result by the number of assessed organizations (a deficient practice may have been noticed more than once in a given organization, since the same practice may have been the source of several potential problems)."

Think about the criteria he uses to define deficient practices, one that I would say is liberal and forgiving to software organizations. No one would or should purchase a product that reflects mal-practice by those who build it. I am not in favor of lawsuits, but lawyers would have a field day with this data.

We as users of software may be part of the problem. We are too forgiving when we encounter defects. It is simple to reboot when a problem on a personal computer (PC) occurs. It is a nuisance or sometimes worse, but we reboot and press on. We all know when a new release of PC software hits the streets we will encounter defects. The suppliers even admit there are defects. Yet we users buy because we want the new function. We can't wait, and in turn we motivate poor practice by the suppliers. We need to become more demanding and with some suppliers this is an absolute must. The suppliers need to learn how to produce better software solutions, because it is possible. They can start by reading and using books like Poulin's.

As a consumer, I implore all software suppliers to put this book to good use. We might live with unavoidable PC reboots, but how would

---

* CMM and Capability Maturity Model are registered at the U.S. Patent and Trademark Office by Carnegie Mellon University.

you feel if the pilot on your next flight announced "Hold on, I need to reboot the flight system"?

Let me restate Poulin's set of practical practices. All of these are quite easy to put into use and are not difficult to understand:

- Understand the requirements before you invest too much in building the wrong solution.
- Plan the project at a level of detail that provides further insight into how to deliver the required solution.
- Measure and track the project at sufficient detail to ensure the project will be delivered as required.
- Ensure that all on the project understand the quality objectives and then assure that these are being met.
- Control project assets, especially changes that always happen during the project life cycle despite what some managers hope for.
- If you need to subcontract, manage the subcontractor to ensure you get what you want, when you want it, and how you want it.
- Provide appropriate process understanding, tools, and methods to the project team members so they can transform the requirements into the desired deliverable solution.
- Ensure that those who need to know are kept informed as the project evolves from start to finish.
- Admit that practitioners make defects and know that high percentages of defects can be removed before turning the work over to others on the team who are dependent on its quality level.
- Protect your customer and their interests at all times.
- Learn the value of defining processes critical to the project and keep these definitions as simple as possible, but define them so there is agreement.
- Prepare the project team members with training to enable their capabilities to deliver the best of possible solutions.
- Build a common cultural understanding of what the organization holds as values for delivering solutions to customers.
- Understand that risks will occur on projects, but risks can be evaluated and managed, so they do not become problems. Read Poulin's book to get a better understanding of what these practices require. These are not complex ideas to understand and they are not complex to implement. Poulin provides more detailed understanding of these in his book. If you cannot start with all the practices then at least start with some. My own personal favorites are:
- Reviews and Inspections, as these are the least costly to implement and have the greatest economic and customer satisfaction return.

- Training, as our industry is constantly changing, and even the best people on our projects must be enabled to demonstrate their capability to deliver; training enables these capabilities; training is a necessary precondition to best work.
- Building a Culture, as a team that understands its purpose and objectives will find a way to use the best practices in our industry.

I do not know anyone who comes to work to deliver poor products, to make defects, or to overrun costs on projects. People take pride in building something good, but sometimes the system gets in the way. The system can be changed but it often requires help to change. This book can help you change the system in your organization. I invite you to learn and understand how you can do better on your projects. I know you all can. I've seen it many times once the organization and managers decide that there is a better way. Poulin's book can enable you to do your best and find that better way.

Ron Radice
November 4, 2004
Andover, MA
www.stt.com