Menlo Innovations

Secrets of Software Success:

Adapting Projects to an Accelerated Society

Richard Sheridan © 2004
President,
Menlo Innovations

Menlo Innovations
212 N. Fourth Ave
Ann Arbor, MI 48104
Phone (734) 665-1847
Fax (734) 665-2990
www.menloinnovations.com

The Crisis in IT

Like manufacturing in the 1980s, IT is now threatened by a crisis in quality and foreign competition. In 2003 alone over 4 billion US dollars was spent in India on IT services[1]. Although this is only a small percentage of current IT spending, history clearly shows the number will grow astronomically over the coming years. If we want to survive and thrive in the coming years we must learn how to resolve the crisis in IT and develop software successfully in an accelerated society. This paper introduces a new approach to software development and describes why it is required to build software successfully in the 21st century.

[1] The Economist, March 2003

A Team that Rocked

I've spent over 25 years working in the software industry and have actively participated in software development initiatives as a developer, team leader, architect, manager and executive.

In my previous role as Vice President of Software Development I had a singular goal, to make a software development team that rocked. I wanted the best software development team in town and I was determined to make the effort to get there. I needed a team that was agile and adaptive; a team that was responsive to very real business needs, needs that changed frequently and grew aggressively as the business itself changed and grew.

I have built that team. And I have repeatedly helped others build similar teams. I have seen the dramatic impact of changing not just the development tools but the entire development culture. We literally build teams that rocked, and you can too. Along the way I learned an interesting lesson: if you really want to be competitive in this marketplace, be prepared to change everything.

Common sense fails.

Conventional wisdom fails.

If you want to dramatically improve your results then be prepared to embrace dramatic change.

A New Metaphor for Software Development

Dr. W. Edwards Deming was a 20th century American statistician. At the encouragement of the U.S. government he visited Japan after World War II to teach Japanese industry how to improve quality through the application of statistical methods. As history demonstrates, Japan learned Dr. Deming's methods and applied them well.

After it became evident that Japanese industry was more successful at applying his techniques than American industry, Dr. Deming wrote a book for American industry called "Out of the Crisis." It was written to the managers of American manufacturing firms and taught managers a new way of thinking about the crisis they faced. Dr. Deming taught about variance and quality. He outlined 14 points for how to manage companies effectively. He taught that quality cannot be inspected into a product[2] it must be an attribute of that product.

[2]. Deming, W.E. (1986). *Out of the Crisis*.

These simple principles of variance and quality, the foundation of modern manufacturing quality practices, are basically unknown in the IT industry. Dr. Deming's insights into manufacturing, that heralded the rise of modern Japan, are not applied. His fourteen points for the management of quality[3] are neither studied nor implemented. This is unfortunate and must change; for treating software engineering as a manufacturing discipline is perhaps the best way to solve the software industry's crisis in quality and productivity. And make no mistake, we are in a crisis.

Project Failure Rates

Each year the Standish group publishes a study of projects in our industry. The numbers are not pretty. According to The Standish Group Chaos Report in the year 2000 almost a quarter of all software projects were cancelled outright wasting over 67 billion investment dollars. Another 21 billion were spent on cost overruns. Incredibly, even for systems that deliver, 80% of the budget is spent repairing flaws the team itself introduced into the software[4].

The rate of failure is so high and the variation so great that the success or failure of any given project is, to most managers, functionally random. It is not surprising that sponsors are reticent to support software development initiatives when reputations and possibly careers are exposed to unknown forces.

If we want to survive and thrive in the coming years we must learn how to resolve the crisis in IT. But be forewarned, to deal with the crisis you will have to change everything you currently do. And even change is not enough, as Dr. Deming made clear years ago, you cannot simply copy my success. You must develop profound knowledge about what we do and why it works. Only then can you achieve for yourself the results we achieve at the Menlo Software Factory. Only then will you be ready for the challenges in IT in the 21st Century.

A Manufacturing Discipline

I propose a new metaphor for software development; software development as a *manufacturing discipline*. Software Engineering as a manufacturing discipline differs from conventional software development to a similar degree that a well managed manufacturing facility differs from an eclectic collection of craftsmen building handcrafted products.

Today, software development almost universally operates in the craftsmen model. The Software Factory seeks to move us to the manufacturing model. Although both models seek the same objective, a working product, the processes employed to achieve the goal are strikingly different.

---

[3] Ibid.

[4] MIT Magazine of Innovation – Technology Review, July/August 2002

Ironically, highly productive manufacturing practices initially strike the craftsmen as inefficient. This is because a manufacturing model **does not seek to optimize work around an individual worker** but around the entire production facility.

For example, several years ago I observed a worker stamping out parts at the Grand Blanc Stamping plant near Flint MI. This worker was working at a moderate and maintainable pace, but clearly not as fast as he could. At first glance the casual observer may conclude that the operation was inefficient. The worker could be working faster and stamping more product in the same amount of time. But, once you understand the whole operation, it is obvious that optimizing around a stamping operator is not a proper objective. In fact, that worker working faster and producing more parts would cost *more* than it would save as the practice causes a great deal of waste.

There are several reasons for this; first, the rest of the plant only needs a specific number of parts. The plant needs to store the excess if they cannot be used immediately. This is a cost. If the worker made a large number of excess parts, they may need to be moved off-site to a warehouse; moving the part is an additional cost. Worse, the pre-stamped parts may cease to be necessary at some point in the future so, all of the excess parts in the warehouse suddenly become scrap making the original production and the scrapping process additional costs. Suddenly it becomes readily apparent why the stamping plant shouldn't optimize around the individual stamping worker.

Sub-optimization, optimizing around an individual or activity and not the end work product, is detected and eliminated in a manufacturing model. This is counter-intuitive to craftsmen and hence counter-intuitive to the software industry.

A manager wishing to remove randomness from the equation of software engineering and build a manufacturing discipline will have to abandon most of the currently accepted but fundamentally incorrect software development practices.

Shifting Focus

Large companies often suffer the most in delivering software projects successfully. They have the wrong focus in how they create and build their software teams. When a new project is started, a completely new development team is assembled to deliver that project. Project goals, schedules, requirements and budget are established for that specific project in terms of a project plan. The engineering staff is dedicated full or part time to the project to deliver the requirements as identified in the plan.

It sounds reasonable; the activities are all important and required. The problem is the focus isn't on building a functional team that knows how to deliver projects. The focus is on building the project. At project termination, when the product is either delivered or abandoned, the team is abandoned. The team is either

destroyed (reassigned as individuals to multiple new projects or discharged) or maintained (kept as a team and given a new project). Neither answer is sufficient.

The project centric focus in team building is a problem, not a solution. It has been tried and it continually fails. It is based on the craftsmen model and actively works against our ability to deliver software successfully.

I propose a new focus. Think of the software delivery team as a working factory; a factory that knows how to deliver a specific product - working software. The focus for the programmers is doing the correct practices to keep the factory working. The focus for the managers is creating an efficient factory process and training the workers to follow that process properly. If the factory managers have built a proper factory they have confidence that the work product ultimately produced will be of satisfactory quality to the customers. The factory managers concern themselves with the operations of the factory; with continually improving the development process, not with finding and putting out fires.

A given software factory is likely to be staffed with anywhere from 5 to 50 factory workers. The workers may be building just one project or, preferably, up to a dozen projects at a time. Of course, some people in the factory will be concerned with the specific concerns of a particular customer; they will be dedicated to working with that customer and their associated users to ensure the end result meets the customers' needs.

However, most of the factory workers operate in the context of producing product (working code) with little concern for the specifics of which project they are working on. In fact, one of the great strengths of the factory is that workers seamlessly move from project to project allowing factory capacity to easily scale to customer demand.

Software Factory Characteristics

A Software Factory differs from craftsmen custom software projects to a similar degree that Ford's first automobile assembly line differed from hand-crafted automobiles. Both seek the same objective, a working product, but operationally the process employed to achieve those goals are strikingly different.

To develop software successfully you must change focus:

Old Focus	Correct Focus
delivering projects	delivering business value
obtaining a sufficient budget	achieving return on investment
getting the entire budget approved	delivering the most important features next
getting the right staff and skills	creating the right process
getting the stakeholders' approval	achieving the stakeholders' participation
getting the requirements right	getting the users' feedback
dedicates resources to specific projects.	optimizes resources across multiple projects
developers doing all roles	forbids developers from performing contradictory roles

I am serious when I say that in order to be competitive, all of your practices will have to change.

The Context Feeding the Crisis

To appreciate why one approach to software development succeeds while another approach fails it, is important to understand the context in which our industry operates. Understanding the context helps answer many of the critical questions around software development projects, including:

- Why are current software projects so unproductive?
- Why are intuitive fixes so often wrong?
- How should our projects change to better fit our times?

Clearly, something is going on around us that causes our typical approaches to fail. Context is everything.[5] The context for developing software today is the context of an accelerated technological society. Technology is advancing at an unprecedented rate. Things are possible today that weren't even conceivable to most people just five years ago. The world has seen more technological innovations in the past fifty years than in all previous years of human history combined. There is no sign that the rate of change is letting up. Recently, we've seen our society embrace and adopt the following significant new technologies:[6]

- Personal computers were adopted by ¼ of the U.S. population within 16 years.
- Cellular telephones were adopted by ¼ of the U.S. population in 14 years.

[5] Larry Downes and Chunka Mui presented these core principles in "Unleashing the Killer App."
[6] National Center for Policy Analysis, http://www.ncpa.org/bg/bg147/table2.gif

- The Internet Browser was adopted by ¼ of the U.S. population in 7 years.

Major technological advances are occurring at a phenomenal rate. You cannot find a teenager today who remembers doing a research project without the Internet.

It is important that you understand the context of an accelerated society. Understanding the rate of change helps you to develop a real understanding of why the factory approach to software development succeeds.

During the dot com phenomena, two principles were extensively hyped and taught. The principles were used to hype excessive stock prices. It was an incorrect application of the principles. In fact, the principles worked against the dot coms, in the same manner that they work against your business and in the same manner they work against any business that must develop custom software.

The two principles we use to capture the nature and impact of accelerated technological change are:

- Moore's Law
- Metcalfe's Useful Equation

I use these principles as a metaphor for the rate of change of technology around us.

Moore's Law

Moore's Law was an observation by Gordon Moore, co-founder of Intel, on the rate of growth of fundamental computation power. Practically speaking, in 1975 you would have paid $32,000 for computational ability you can buy today for $1. In the world of computers, the golden triangle of *faster, better and cheaper* is actually true. This principle has been seen in multiple areas including network bandwidth, storage capacity, and wireless coverage.

Metcalfe's Equation

Dr. Robert Metcalfe's equation describes the network effect. The more nodes on the network the more valuable the network becomes. Metcalfe described that value as the number of users squared. Networks such as telephones, fax machines, email and web browsers reach an apex point after enough people have adopted the technology that their value is unquestionable. Unfortunately for software developers this apex is being reached faster and faster. Don't make the mistake of believing it is over, there are more breakthroughs like telephones, faxes, and Internets on the horizon, you may not see them yet, but

history clearly teaches use that they will soon be here. In some cases, they are already here. Time will tell which innovations will really catch on.

Moore and Metcalf as Metaphor

We think of Moore's Law and Metcalfe's Equation as metaphors for the rate of change of technology around us. They are simply the latest installments in a technological juggernaut that began over 500 years ago, since Gutenberg invented the printing press in the 1440s. New technology causes disruption. It disrupts how governments, business and society operate. The first book off of the Gutenberg printing press was the Bible. It is not a coincidence that the Protestant Reformation occurred within a generation. New technology causes disruption.

The last 50 years have been technologically accelerated times. The invention of the transistor and the fast development of modern computational capability leave other parts of our society far behind. Technology changes faster than society, society changes faster then business, and business changes faster than government. This gap between technology change and other aspects of society may be thought of in many ways. We like to consider the gap to be an opportunity gap. It presents an opportunity to provide your customers with additional new products and services. However, the gap is also a disruption gap. It disrupts existing businesses and business processes potentially making them obsolete. It has certainly made most conventional software development practices obsolete.

The Pressure Cooker

Moore's Law, Metcalf's Equation and our accelerated society directly impact our ability to develop software effectively. The pressure cooker in which software is developed includes:

- rapidly evolving new technology.
- rapidly changing networks.
- new and emerging business models.

This pressure cooker impacts our software projects in many ways. Let's list a few:

- **New Hardware Platforms:** Software developers are typically writing for new target hardware platforms every eighteen months.
- **New Languages:** Software developers have faced four major language changes over the past fifteen years alone.

- **New Developer Tools:** Software developers usually work with a tool for less than twelve months before the tool is revised or replaced.

- **New Developer Techniques:** Tricks and techniques that made a developer a master yesterday may no longer be applicable or useful today. Even worse they may be detrimental.
- **New Demands:** Developers are faced with continuing increased complexity in requirements. In the past fifteen years we have seen demands for graphical user interfaces, networks, client server architectures, distributed systems, browser based software and wireless. Each addition increases the complexity of software and compounds our quality crisis.

The Problem with Complexity

How complexity effects quality can be seen in the following simple fictional example.

Say in 1980, you released a new operating system for a brand new personal computer. We will call the operating system SOS, an acronym for Simple Operating System. At that time a simple operating system may have consisted of only 10,000 lines of code. If you had a defect density rate of one defect for 1000 lines of code SOS would have shipped with 10 noticeable bugs; a problem, but one that could be overcome with a few incremental releases.

Now jump forward 24 years. You release SOS-2004. SOS-2004 has a graphical user interface, supports networks, file sharing, Internet access, Internet browsers, scanners, printers, cameras, CDs, DVDs and 3,000 other different devices. S0S-2004 is no longer simple; it has over forty five million lines of code. If you have a defect density rate of one defect for 1000 lines of code you have shipped your new operating system with 45,000 defects. You now have a quality crisis! If you are 10 times better in 2004 in removing defects than you were in 1980 you still have 4,500 noticeable defects. If you are 100 times better in 2004 in removing defects than you were in 1980 you still have 450 noticeable defects. With 450 noticeable defects, you still have a quality crisis. It is an utterly unacceptable number. The problem is, as an industry, we are nowhere near being 100 times better removing defects today yet *we are* producing tens of millions of lines of code. And the defects in commercially available operating systems and applications are all too readily apparent.

Our software development process must adapt to this pressure cooker or it will be crushed by it. The pressure cooker is our reality. It is the world in which we develop software. It is not enough to simply learn how to live with the pressure. We must learn how to change the pressure from a negative force into a competitive advantage. We must learn how to embrace change.

Common Problems

Almost ever team I visit is at first embarrassed to admit their problems. They think they are unique. They are not. The problems you are having with your development initiatives are the same problems I see at every company I visit – and the numbers are now in the hundreds. These problems are:

- Project is late.
- Project is over budget.
- Software is buggy.
- Software doesn't meet user expectations.
- Software specifications are changing too often.
- Little evidence to management that the project is progressing.

On-Schedule, On-Spec and On-Budget

A common desire expressed by software development managers is "I would be happy if I could implement a development process that delivers software on-schedule, on-specification and on-budget." This is an admirable goal, unfortunately it is not nearly enough. The statement contains significant assumptions that must be true to be successful:

Assumption 1: The schedule delivers the software in time for business goals.
Assumption 2: The specification describes the right features.
Assumption 3**:** The budget investment provides a positive return.

It is possible to develop software on-schedule, on-spec and on-budget and still fail miserably as a business. Here are seven ways:

1. Deliver on-schedule but before the market is ready for the product.
2. Deliver on-schedule but after the market is interested in the product.
3. Deliver on-schedule but after the competition beats you to the market.
4. Deliver on-specification but with features the market doesn't need or desire.
5. Deliver on-specification but with a user interface that is too complex for the users to use successfully.
6. Deliver on-specification but with features that the market doesn't understand.
7. Deliver on budget but the project still loses the business money.

It's fun. It's easy. See if you can add to the list other ways you can be schedule, on-spec and on-budget and still fail miserably. So, if the expression above is inadequate perhaps it would be beneficial to replace it with a better one, consider:

> **I want to deliver software to meet the business needs when the business needs it.**

This could be taken as yet another meaningless "business mission" statement. It is not. Connecting the software developed to meet true business interests is at the heart of what makes a software development process successful. This is how to please your customers and develop outstanding references.

Technologists say "Great. Tell me what you want and when you want it and I will build it for you. But don't change your mind once I get started or we will never finish." Unfortunately, it is not that easy. Business doesn't necessarily know what they want or when they need it. Technologists need to work with business to help business discover what it wants and what it needs. Business is under the same dynamic forces of the pressure cooker. Business lives in an accelerated society and has to make strategic decisions that later will be changed.

In business, "what I want" and "when I need it" is a dynamic conversation. The answers may very well change tomorrow. Technologists must be able to provide business with valuable software in light of the very real business environment that surrounds us.

Most organizations are good at identifying how or why new development practices "won't work here." Changing long term organizational habits and operational practices is difficult and often painful but it may be the only way to survive. Before finding all of the ways new practices "can't work here" consider the following questions:

> How are your current practices working for you?
>
> Is your business getting the results it needs from software development to thrive in today's economy?

Depending on the mood we are in we may ask this trick question:

> How good of a job do you think your team is doing developing software? Are they average? Above average? Below average? By how far?

If you answer average or a bit above average you have fallen into the trap, perhaps deservedly so. In our business you need to be performing way above average, an order of magnitude above average, because *average* is really, really bad.

As stated previously, it has been calculated that on average, 80% of the budget for custom software is actually consumed removing bugs the developers themselves introduced. The rate of failed projects and rework in our industry is almost

incomprehensible. You can be better than average and you can still:

- Fail on half of your large projects.
- Restart half of all your projects, at least once.
- Waste, statistically speaking, over 85% of your development dollars.

Clearly being average is not enough. Being better than average is not enough. To survive and thrive in this field, you need to be exceptional.

The Typical Responses to the Crisis

Many organizations have realized for years that they are not getting an adequate return on investment for their software development dollars. They know that many of their software initiatives are failing miserably. Many have attempted to solve these problems by moving from an ad-hoc to a formal process. Typically, the new processes are implemented with little or no success. We encounter these new processes so often that we have given them special names, we call them the "Soviet Solution" and "Magic Bullet Technology."

The Soviet Solution

A common solution I often see to the software crisis is what I call "The Soviet Solution." In "The Soviet Solution" management ascertains, often correctly, that in previous projects, requirements have gotten completely out of control. Therefore, management seeks to implement a new centrally controlled process to capture and control requirements. This typically involves a central committee I call The Supreme Soviet. In the Soviet Solution, the worthwhile objective of getting control of the requirements leads to the implementation of the following remedies:

Remedy 1: Getting the Requirements Right the First Time

Getting the requirements right the first time is a natural first response to software development process problems. The goal is to produce up-front a complete set of "correct" requirements. The requirements typically take the form of a large document or a set of documents and a long list of features. More enlightened teams may produce this document as a set of use cases. Less enlightened teams produce more ambiguous requirements such as *the system must be scalable* and *the system must support multiple languages.*

A large amount of time is allocated at the beginning of a project to create this set of requirements. Again and again I have met teams that have spent a year or more assembling these requirements without writing a single line of code. It is believed by these teams that if the requirements can simply be captured correctly the first time the most difficult problems with software development

will be solved.

The developers may even support this approach. They may provide anecdotes about the amount of time wasted on previous efforts involved in changing code as the requirements changed.

There are many problems with this remedy. It has been tried many times and experience clearly demonstrates it doesn't work very well. The reasons it doesn't work are subtle. The subtle nature of these reasons is why we set the context of an accelerated society earlier.

Failure Point 1: The Target is Moving
The context for any project is the accelerated society we live in, remember Moore and Metcalfe. If a project spends a year *getting the requirements right* it is almost guaranteed that many of the assumptions on which the requirements were founded have already changed. A year is a very long time. Yet we frequently encounter projects that spend a whole year gathering requirements. Of course, during the year spent gathering requirements the world continues to change and the requirements gathered at the beginning of the year are out of date by the end of the year. Typically the large requirements document produced by these initiatives end up in the garbage heap.

Failure Point 2: It Is Almost Impossible To Get Requirements Right
Even if everyone including users, stakeholders, developers and executives agrees that a requirement or feature is right, does it mean that the requirements are right? It is difficult to evaluate a requirement when it is written because it is very difficult for most people to envision a working system from a document. How does the system really impact the user? Does it help? Does it hurt? Is, in practice, the feature irrelevant? Experience clearly demonstrates that until a *real* user runs a *real* system in a *real* environment to accomplish *real* work, the correctness of the requirements are, at best, a guess. At worst, they are a poor guess.

Spending a large amount of time getting the requirements right without actually putting a working system in front of real users is a guaranteed way to get the requirements wrong. Business is complex. Computer systems are complex. Trying to understand how business and computer systems intersect in a real environment without installing and running the systems is practically impossible. Catch 22: If you do all of the up front work to understand the requirements, the amount of time spent getting there has likely moved the project back into the *moving target* problem.

The chicken and the egg problem of creating software and requirements is very real and very deceptive.

Remedy 2: Executive Signoff and Control

A second common remedy I frequently encounter is more central control. Executive signoff is required from multiple department heads for "go" or "no go" decisions at each phase of the development process. Surly, all this review is making the project better? After all, the large requirements document is filled with multiple signature pages.

Unfortunately, this remedy assumes too many things. It assumes:

- The executives understand the system to be built.
- The executives understand the requirements.
- The requirements have been written in a manner that facilitates "go" or "no go" decisions.
- The political environment equips executives to make hard decisions.

This remedy doesn't work because it doesn't address the real problems facing development. It doesn't provide executives with true control points. Proper management is about more than "go" or "no go" decisions. A software project is always about a series of trade-offs. Trade-offs between scope, schedule, budget, and a project's return on investment. For executives to be meaningfully involved in software development initiatives they must be able to address intelligently all of these variables. A sign-off sheet in a large requirements document is simply a political control and is used to assign blame, not fix problems. A sign-off sheet may be needed for work authorization purposes, but let's not fool ourselves into thinking that it gives management any meaningful controls.

Remedy 3: Resist Changes To Requirements

After the requirements have been signed-off in the large requirements document any change to the requirements is actively resisted. Executive permission, or a specific change board, is required to change anything.

Of course, everything we have studied about our accelerated society tells us that this response cannot work. We don't get requirements right the first time. Even if we did, requirements that are a year or more old are already irrelevant in an accelerated society. A meaningful development process must learn how to embrace change, while delivering real value.

The software industry has been trying these three remedies for over 20 years without significant improvements in results. Perhaps more technology is the answer?

Magic Bullet Technology

The Magic Bullet solution is not so much a solution as a mindset. This mindset believes that all of the software development problems encountered in the past will be eliminated by "*some new technology just on the market.*" The technologists are often the champions of this type of thinking but management is certainly not immune.

The magic technology may take many forms:

- A new software development environment or tool.
  (Objects, Java, COM, XML, Application Servers, J2EE, DOT NET, etc.)

- A software analysis, design or requirements tool.
  (Case Tools, Rational Rose, UML Drawings)

- Project management software.
  (Microsoft Project, Rational Suite, etc.)

The Magic Bullet Solution never worked. This is not to say that new technology doesn't sometimes bring some benefit, but it will never be *the answer* that single-handedly saves your projects. Technology can't save your projects because research clearly demonstrates that technology is not the limiting factor in delivering software successfully. For almost all software development projects the selection of a specific technology (C++, Java, C#, Application Server, etc.) and the technical skill level of the developers is almost irrelevant to the software project succeeding or failing.

The Soviet Solution and the Magic Bullet Solution are intuitive, rational, and well intentioned. They are also wrong. We know they haven't been working because in places where they have been adopted there is no significant improvement in the project success rate. These approaches continue the cycle of failure.

The Software Factory Response to the Crisis

The needed response to the current software crisis is far different from the typical response; the needed response is for business-centered, creative, and adaptive teams using agile processes that allow them to produce exceptional quality while embracing change. A common definition of agile software development is:

- Features may be added in any order.
- Features may be released at any time.

This is a good start. In The Software Factory we add to this definition the following:

- Software development is driven by business values not technical values.
- Business must have a clear mechanism to understand development progress.
- Business must have a clear mechanism for directing development.
- Quality must be intrinsic to the development process and not rely on massive inspection.
- Knowledge must be communicated continually.

Being agile is key to being successful for your team and for your entire organization. I have worked with companies both large and small, in the private and public sector. Thus I can assure you, change is difficult. It is also absolutely essential.

Next Steps

The change required to build a truly agile team is significant. The time to begin is today.

I have produced two additional resources to provide you with more information on building and implementing a Software Factory. The first is an additional white paper entitled:

> Secrets of Software Success – The Nature of the Team
>
> If you registered for this paper: Adapting Projects to an Accelerated Society then you are automatically signed up to receive this next paper.

The second is a free seven part mini course entitled: Eliminating Waste in Software Development Teams. To sign up for it go to the Menlo Innovations website at: www.MenloInnovations/method/index.htm

Also, consider taking one of the many Menlo training classes on software development and the Secrets of Software success.

Menlo Innovations Training

Menlo provides multiple training classes for teams seeking to improve their performance. The classes listed below are specifically recommended to those interested in building and operating a Menlo Software Factory. Please give me a call at (734) 665-1847 and we can discuss your needs in more detail. Ask for Rich Sheridan.

Secrets of Software Success – 1 day

> The elements of successful software initiatives go beyond "best practices and tools." Business and technology are inseparable. Customers and business are inseparable. A successful software initiative must fuse customers, business and technology into a single functioning team: this course teaches how to build that team.

Software Project Management for Agile Teams – 2 days

> This class teaches how to manage an agile development team by simulating all of the activities found in a typical development iteration. It also teaches how to change how the development team interfaces to the rest of the organization in order to fully achieve the full benefits of agile development; flexible software, responsive teams and timely software releases.

High-Tech Anthropology 101 – 2 days

> This course teaches how to effectively capture requirements using the concept of High-Tech Anthropology™. It will teach you how to design software that your users will love. Participants practice writing use cases and performing other activities required for building high-value, user-friendly applications. Additional topics include: interviewing, job shadowing, persona writing and creating storyboards.

Additional information on these and other Menlo courses is available on our website.

Menlo Innovations LLC
212 North Fourth Avenue
Ann Arbor, MI 48104
Phone: (734) 665-1847
Fax: (734) 665-2990
www.menloinnovations.com / rsheridan@menloinnovations.com