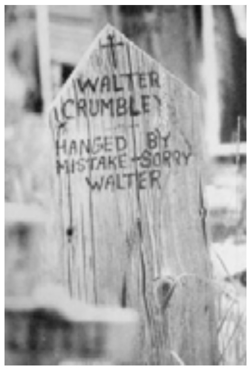
The Agile Method and Other Fairy Tales

By David Longstreet

We find that whole communities suddenly fix their minds upon one object, and go mad in its pursuit; that millions of people become simultaneously impressed with one delusion, and run after it, till their attention is caught by some new folly more captivating than the first.
- Charles Mackey – 1852, (*Extraordinary Popular Delusions and the Madness of Crowds*)



A few years ago I took a photograph of an old wooden tombstone. The tombstone read, "Walter Crumbly, Hanged by Mistake – Sorry Walter." It seems like the software industry wants to collectively hang the waterfall software development method and structured methods. In the not too distant future the software development industry will wake up and realize the move to Agile Methods was a mistake. This is not the first time we have seen an entire industry wrong. The Y2K issue was greatly overblown and the dot com boom was another black eye. The entire software development industry has a credibility gap.

Background

For over two decades of my life, I have been dedicated to the idea of improving software productivity and quality. I have traveled the globe consulting and studying software organizations that support just about every industry including banking, aerospace, retailer, animal food, telephone, consulting companies, healthcare, defense contractors, package delivery, automotive, travel, government agencies, and insurance. I have worked for organizations with only a few employees and others with billion dollar budgets.

Over the years, I have worked with government agencies and private companies in Europe, Asia, and the USA. I have consulted with organizations that create software to launch rockets and organizations that manufacture animal food. I have worked with just about every industry that uses software.

I have consulted on every continent except Antarctica. The circumference of the earth is about 25,000 miles, so I have circled the globe over 120 times collecting nearly 3 million frequent flyer miles. Over the years, I have been on a quest to learn and understand software development and to move the industry and my clients forward. I have learned a tremendous amount from my clients. I have had the opportunity to work with the best in class and the worst in class software development companies. What I want to do is simple. I want to bring a level of professionalism to software development.

Over the years, I have not limited myself to just studying the software industry. I have tried to retrace the footsteps of other industries such as architecture, engineering, and medicine in hope of learning from their successes and failures. I have interviewed music conductors, architects, industrial designers, graphic designers, and others in the hope of gleaning information useful for my clients and the software industry as a whole. I reject the notation held by that some that software is somehow unique. The more I study the problem the more I am convinced software is like every other industry.

I have not limited my studies to just industry. I am an active member of the Academy of Management, the professional association of those who teach management at universities and colleges around the world. I participate in conferences and referee articles with the intent of understanding and sharing the best management practices with my clients.

I have not always been a consultant. I started this software journey as a programmer. My first software language was machine instructional cross assembler. It was basically hexadecimal machine language. I could really pump that stuff out. Early in my career I wrote FORTRAN code for theoretical physicists solving complex mathematical equations. Over my career I programmed in COBOL, C, C++, and Java. I actually co-authored a DOS 5.0 book that is obsolete now. Prior to becoming a consultant, I managed software developers, testers and production support teams. I have managed both technical staff and functional groups. I started out with punch cards and now I use a Mac.

You can read more about me at www.SoftwareMetrics.Com/client.htm and about me at www.SoftwareMetrics.Com/david.htm.

Once Upon a Time

…. There was a doctor, Ignaz Semmelweis of Vienna Austria, who believed statistical methods, could be applied to medicine. He believed a lot could be learned about disease could through systematic study and by observation. The spread of a fever and infections was a real issue for hospitals during the mid 1800's. Dr. Semmelweis discovered doctors were the primary cause of the spread of "the fever" and infectious diseases in hospitals, in other words, physicians were causing their own problems. In the mid-19th century it was common for a doctor to move directly from one patient to the next without washing his hands, or to move from performing an autopsy on a diseased body to examining a living person. Semmelweis hypothesized that "particles" introduced into the women caused puerperal fever, and that these particles were spread on the hands of the doctors and students. His contemporaries mocked him and said these little particles (bacteria) were a figment of Ignaz's imagination.[1]

His ideas regarding the application of statistics to medicine, and especially his idea that physicians caused the spread of disease, were rejected by many of his contemporary physicians. Semmelweis was eventually fired from his job as a physician in Vienna. At a conference in 1861 Semmelweis presented his ideas and most of the other speakers rejected his theory. Many doctors argued even if Semmelweis was right it would be too much work to wash between patient visits.

[1] Horne RA. "Science Fraud--a Review of Broad and Wade: Betrayers of the Truth: Fraud and Deceit in the Halls of Science." Physiological Chemistry and Physics and Medical NMR15. 5 (1983)

Like Ignaz Semmelweis through years of systematic study and observation, I have come to the conclusion that software developers cause most of their own problems. The root cause of most of the problems facing software development is actually caused by software developers themselves. They are creating their own complexity and chaos.

Like physicians 100 years ago, the real angst is that software development causes most of its own problems. You don't need to become Agile, you need to fix your problems. By systematic study and observation you can learn about your organization and be able to differentiate between self-inflicted problems and those problems you can't control.

You may be thinking "Function Points" are a figment of my imagination. Not long ago, I was accused of practicing some sort of software voodoo. I have heard many times you can't measure software or the process; it is too complex, and it changes rapidly. This idea of unpredictability and complexity has been used in physics (study of planetary motion), medicine (how infections spread in hospitals), and geology (plate tectonics). Throughout history this argument has been used many times to squelch the idea of the application of the scientific method. The argument goes like this, "my subject area is too complex and unpredictable, so the scientific method does not work. I can't plan ahead, I can just respond."

Agile methods want continuation and formal acceptance of the status quo. The new idea is not Agile. The new idea is the application of the scientific method, measurement and structured methods to software development. Up to this point in time software development has been a Wild West endeavor. Many people have an investment in the past and are unwilling or unable to change. IT has been sloppy. There is nothing new with Agile, because it only tries to formalize sloppiness. The IT industry has never been known for its disciplined approach. The IT industry has a history for missing estimates, a history of Y2K, a history of the dot com, and a history of horrid customer service. The IT industry is best known for scrambling around trying to find its collective way. I am not asking for a continuation of the status quo, I am asking the IT industry to mature.

What I am proposing is simple. I am bringing a level of professionalism and rigor to the software industry, and I hope you join me.

There is little disagreement among software professionals regarding the industry needing to improve. There is violent agreement that the software industry is plagued by changing requirements, poor estimating, weak communication, and disorganized outdated documentation. The Agile camp has thrown up their hands and believes this is the way it is and it is the nature of our business. They have decided it is best to simplify (become Agile) to navigate through this sea of chaos. If the chaos is outside our circle of influence and we are a victim of circumstances, then there is nothing that can be done except become Agile.

Requirements Raining

An Agile proponent will argue there is limited value in requirements specifications because the requirements are ever changing. You need to be “agile” because requirements just change. To believe this you have to believe every industry, insurance, travel, investing, banking, telephony, medical that interfaces with IT cannot provide clear and complete requirements. It seems like every industry can’t communicate requirements effectively to the software industry.

It is like a friend of mine, Wes, who has been married and divorced 7 times. He was complaining about “all women” and I made the bold suggestion perhaps he was the problem and he should seek some professional help. To my surprise he took my advice. A few months later I ran into Wes and he told me he was already on his 3rd counselor. Some people just do not learn they are the problem and I believe Wes is one of them.

The problem with raining requirements is related to those in the software industry, not those who are providing requirements. It is not all those dumb users, but those in software development. Instead of fixing the problem of eliciting requirements the software industry moves onto some new methodology like Agile just like my friend moving onto another counselor.

Study Thy Customer

Tom Kelly is an Industrial Designer for one of the most successful design firms IDEO. He writes in his book *The Art of Innovation*, “Your customers may lack the vocabulary to explain what’s wrong and especially what is missing.”[2] One has to start to ask the question, “If your customer does not know what they want or they cannot articulate what they want, then how exactly should the requirements process work?”

I think everyone remembers being a kid at the big Christmas, Thanksgiving, or some other family get together. It is pretty common for the adults to sit at one table and the kids to sit at another table. I always wanted to sit at the table with the adults, but my grandmother told me I could sit at the adult table only if I acted like one.

I was working with a software development group whose core business is financial management for retirees. Let me repeat myself, they are in the retirement business. During one session I asked some developers how many of them subscribe to American Association of Retired Persons (AARP). For those of you who do not know, AARP is the largest organization in the world dedicated to retirees. Not one of developers subscribed to AARP. Worse none of them had ever visited the AARP website. Then I asked what retirement magazines do they subscribe to – the answer was none! How in the world does a software developer develop software for retirees when they have no knowledge of retirees?

---

[2] Kelley, Tom, and Jonathan Littman, The Art of Innovation: Lessons in Creativity from IDEO, America's Leading Design Firm. New York: Currency/Doubleday, 2001, P. 27.

Most software developers have little knowledge about their companies' core business. I am amazed at the number of software developers who cannot list their companies' competitors. This means these developers have never studied the industry and have never studied their competitors. The primary mistake is the lack of study of the customer and the customer problem. It is as if no one ever thinks to visit the customer and study them and ask "what do you want do to do."

Customers are not part of the Agile process. I can't put it better than a few Agilistas. The following quotes come from the Yahoo Extreme Programming Group.[3]

> "Agile software development starts when the programming starts. What happens before that may be incredibly important, but it's not Agile software development."
>
> "It's eXtreme Programming, which implies writing software, to me. eXtreme Helping The Customer Figure Out What They Want is something else."
>
> "I think it's fair to say that customer practices are not addressed in Agile methods."

It is clear that understanding what the customer wants or helping the customer figure out what they want is not really part of Agile, and in turn not part of software development.

This brings me back to sitting at the table with the adults. Today the core business is at one table and software development is at the kid's table. Why would the CIO be invited to the table with the business if they and their staff have little or no knowledge about the core business? Better yet, the software developers have no desire to learn about core business. If software development does not understand the core business, how is it possible for them to understand what functionality needs to be included or not included in any project, upgrade or release?

I have had this same discussion with several different types of clients, and it does not seem to matter if the core business is health care, insurance, or dog food; most people in software development have little knowledge of their core business. Software developers do not specialize along industry lines and most of them have no interest in learning the core business. Since they know little about the business they cannot offer any strategic direction and the organization loses little by outsourcing them.

Beyond Chaos

Agile is based upon the empirical process model. The empirical process model is based upon practical experience, trial and error, and direct observation. An empirical model is developed when there is little theoretical or practical understanding of the event or phenomenon. The idea of empirical model is that you do not know enough about the

[3] I believe it is unprofessional to list names of individuals who post on public message boards. It is also a violation of the rules of most yahoo groups. If you want to know who posted those comments you need only do a simple search.

events to even predict future outcomes. The empirical process is used to explain things like random sea waves and the motion of offshore structures, complex econometrics, metrology and finance. [4]

The Agile proponents incorrectly assume there are no patterns and things are occurring in a random fashion. They incorrectly assume the environment is unpredictable. I ask those organizations who are considering Agile, "have you ever studied your environment? No, really studied your environment. I ask, "Have you studied your customers and their business?" What I do is study organizations with a microscope and an unbiased eye. I help my clients understand their own organization and customers, so they can start to see patterns and predict costs, schedules, and customers.

Perhaps it is the statistician in me, but I do not believe anything is random. Nothing occurs by random and nothing occurs by chance. The results at a casino are very predictable. The casino is always going to win in the long run! I guarantee you those individuals who believe they beat the casino have never studied their results and kept track of wins and losses. It seems like people only remember when they win. The same is true with Agile.

A student in my statistics class did a research paper where she asked those leaving the casino, "Are you up or down?" Over 70% of the people said they were up. If 70% of the people beat the casino, the casino would quickly go out of business. The problem is people only think they won because they do not document and count his or her wins and loses. If you do not measure you will never not if you are making progress. The same is true with Agile.

Software Is Too Complex

Agile proponents invoke the name of Babatunde Ogunnaike. Ogunnaike is an industrial engineer and has a PhD. in chemical engineering. Craig Larman in his book *Agile & Iterative Development: A Manager's Guide* retells a story originally told by Ken Schwaber. As the story goes Ogunnaike and his researchers laughingly told them systems development has so much complexity and unpredictability that it has to be managed by a process control model referred to as empirical. Like so many things with Agile, they either misunderstood or are misrepresenting the idea of empirical process control and especially empirical modeling.

In general, Agile methods promote their interpretation of "empirical" rather than defined processes, a categorization used by industrial engineers. Industrial engineers use a variety of different techniques to build models and develop products. As Ogunnaike writes on his own website, "we develop and employ dynamic mechanistic models; when more data is available than fundamental knowledge, we apply probability theory and statistics for efficient data acquisition and empirical model development."

---

4 International Journal of Offshore and Polar Engineering
Vol. 13, No. 1, March 2003 (ISSN 1053-5381)
Copyright © by The International Society of Offshore and Polar Engineers

The whole idea of developing an empirical model is to help study a problem where there is little actual empirical knowledge or a “defined” theory. The empirical model is not designed to build products but to understand and to help develop a defined theory. Ogunnaike writes in his book *Process Dynamics Modeling and Control* , “Keep in mind the modeling exercise is not an end in itself; it is a means to an end.”

Empirical modeling is based upon actual empirical evidence or observation instead of a defined theory. Often an empirical model starts with a stochastic process. Stochastic comes from the Greek word, “stokhazesthai” meaning “aim or guess.” A stochastic model is used in contrast to probability theory. In probability theory all variables are defined and an outcome can be predicted. In a stochastic model many of the variables are not known and an “aim or guess” has to be made at an outcome.[5] Once actual knowledge is gathered, a defined theory is developed.

The Agile argument is based upon the idea that systematic study does not work for software development. They believe “most software is not predictable.” If something is not predictable, then it cannot be estimated or planned. In their minds, since software is not predictable then software development is basically a random process. This is the rational for using the “empirical process.” In most Agile publications and on most Agile websites, proponents confuse the term empirical *process* with empirical *modeling*. Empirical modeling is used when there is a lack of knowledge about an event or series of events.

The primary reason why it is difficult to apply measurement and understand software organizations is because software organizations are chaotic. Chaotic[6] and random are not the same things at all. Every single time a development project is done, it is done differently. Documents are not cataloged and organized. There is no consistent usage of terms and symbols between projects, within projects, and even within single requirements documents. The development process is not random. It is just a mess.

As a side note, Craig Larman misspells Ogunnaike as Ogannaike in his book. The story of Ken Schwaber and Ogunnaike is retold again and again on websites. What is interesting is numerous websites misspell Ogunnaike as Ogannaike.[7] Obviously the tellers of this story are just lifting it from Larman’s book. The Ogunnaike story is basically, I read a book by this guy, who knows a guy, who met a guy that told him this story. By the time the story hits the Internet it is being told in 5th person.

Empirical Process Control

[5] Beichelt, Frank. Stochastic Processes in Science, Engineering, and Finance. Boca Raton: Chapman & Hall/CRC, 2006.

[6] Chaos theory is a method to understand the order of what seems to be a chaotic process. The main component of chaos theory is systems no matter how complex rely upon an underlying order.

[7] Larman, Craig. Agile and Iterative Development: A Manager's Guide. The agile software development series. Boston: Addison-Wesley, 2004. See page 112.

My house was built in 1836 and it is two shades of pink, blue and white.   Any kindergartener can tell you when red and white are mixed it creates pink. The colors of the pinks of my house are “pink velvet” and “soft serenade”. These colors have very defined specifications. The idea of empirical process control is to adjust the inputs to control for the exact specified output. Ogunnaike writes, “The system is treated like a black box and stimulated by changes in input, so that the response can be observed and analyzed.”

The amount of red pigment in the red paint can vary and this can cause variations in the final pink color.   While the paint is being mixed, the red pigment is monitored, measured and controlled automatically. If the red pigment is more than expected then less red paint is used and more white paint is used. On the other hand if the amount of red pigment is less than expected, then less white paint is used and more red paint is used. The inputs are automatically controlled to produce an exact final product. This is automated empirical process control.

With Agile the final product and inputs are not defined upfront.   They are figured out as the project moves along. With Agile the color of paint is not known, nor are the input colors. Now imagine painting my house using Agile methods. I don’t know what color I want to paint my house so the painter needs to bring all the mixing tools and paint colors to my house.   The painter would paint a bit and ask, “Is this color you want? Then they would paint some more and ask, “Is this the color you want?” If I change my mind they just mix up some new colors and keep painting. Eventually the painters would have to go back and repaint (iterate or code/test) what they originally painted. We would iterate through this process until my house was painted. This iteration would be expensive and time consuming.

In empirical process control the final product is well defined. Since inputs can vary, tolerance levels are specified within some given range. With empirical processes control everything is defined including tolerance levels. Nothing is occurring by some random chance. Agilistas are misusing the term empirical and they do not understand the difference between empirical modeling and empirical process.

Pair Programming

Cheech and Chong, a1970’s comic act, portrayed themselves in one of their skits as Siamese twins that were not identical. They were paternal twins physically connected at the hip. Of course this was impossible, absurd and a very laughable idea. Whenever I hear about pair programming[8] I can’t help but think of Cheech and Chong and the absurdity of the idea of pair programming. It is like asking two artists to paint a landscape together or two musicians to share a musical instrument. It is absurd and laughable.

[8] I have been told that nobody really pair programs and it is unfair to even mention this. There are many Agilistas who claim to pair program over “80%” of the time. To validate this google the term, “pair programming” or search a yahoo group like extreme programming.

The idea is that one programmer writes code and the other programmer stands over his shoulder and watches for mistakes. It is suggested that it is a good idea to have one programmer control the keyboard and the other the mouse.[9] When the first programmer gets tired, they switch positions. I think the two programmers are supposed to touch hands like professional wrestles do in a tag team-wrestling match.

I am not sure what problem pair programming is trying to solve. Most of the issues with software development are related to incomplete requirements, not coding. Typically coding errors account for less than ten percent of the total project errors.

Iteration Code/Test (trial and error)

I often wonder what problem is trying to be solved by iterating in the code/test part of the software lifecycle. In the book, *Hitchhikers Guide to the Galaxy* ”hyperintelligent pandimensional” beings got fed up with the question of the meaning of life, so they built a super computer the size of a small city to answer the question. It took the super computer seven and half million years of calculating the answer to the ultimate question of life, the universe, and everything. The answer was “42.” The problem is the builders did not know the exact question.

The code/test iteration is not much different than 42. Incomplete requirements are the biggest issue facing software development. I guess it is clear to the Agile folks that it is only logical to spend more time coding instead of cleaning up requirements or writing concise requirements in the first place.

They believe trial and error is the best method to gather and communicate requirements. As they say, developing a “working product.” It is clear they skip the study of the customer problem and focus where they are comfortable which is on the code. They hope to solve the customer problem by trial and error. The problem with trial and error is you are going to make a lot of errors along the way.

Weight Loss

Agile proponents believe discipline is not necessary and inhibits productivity. By discipline I mean studying the problem using the scientific method. Discipline is not hanging post it notes around a conference room or writing requirements or project plans on a 3 x 5 index card. Using the scientific method one develops a hypothesis then gathers data to support the hypothesis. To really understand the impact of Agile you would first have to understand current levels of productivity. Like studying the customer, this step is just skipped by most organizations moving to Agile. Nearly all companies moving to Agile do not know their current levels of productivity or quality. I

[9] Martin, Robert C. Agile Software Development: Principles, Patterns, and Practices. Alan Apt series. Upper Saddle River, N.J.: Prentice Hall, 2003. Page 13.

Larman, Craig. Agile and Iterative Development: A Manager's Guide. The agile software development series. Boston: Addison-Wesley, 2004. See page 149.

can't blame Agile for this because most software organizations do not know their current levels of productivity or quality.

Imagine this argument from a pharmaceutical company. We don't have any empirical data, but all the anecdotal evidence suggests the drug is effective; therefore, the drug should be approved and adopted.

The argument goes like this, "my neighbor takes this magic pill and she lost weight!" All you need to do to lose weight is take the Hoodia Gordonii Plus (by the way, this is a real product). Who needs to exercise and eat right? If you buy Hoodia Gordinii Plus right now you get a free Weight Loss Hypnosis CD.

If you don't believe these pills work, just go to ConsumerDietReview or the Weight Loss Institute. There are all kinds of testimonials like the following.

"Within 17 weeks I lost 69 pounds. Our wedding photos were just incredible."

Compare these two arguments

> *Do Weight Loss Diet Pills Work?*
> Have you tried weight loss diet pills? Have you wondered if there some diet products out there that are actually help people to lose weight safely without giving up all of their favorite foods?
>
> *Does Agile Work?*
> Have you tried the waterfall method? Have you wondered if there are some mythologies out there that are actually helping software organizations improve performance without having to give up your undisciplined habits?

There are a lot of Agile software developers who readily proclaim the benefits of Agile, but there are very few customers or the recipients of Agile methods speaking on behalf of Agile. I have searched for those customers to sing the songs of Agile.

Creativity

I have learned a lot over the past decade, but I have learned the most from mergers and acquisitions. Acquiring companies want to know as much as possible about the firm they intend to buy. My clients have sent me as their envoy to study potential acquisitions. I gather both qualitative and quantitative data in order to create a holistic picture of a potential acquisition. Over the years I have taken the opportunity to compare and contrast data from many different companies. Without any doubt the most disciplined organizations are the most productive, creative and innovative. This not only true of IT, but for athletes, musicians, investors and organizations.

Think about this from a personal perspective. Are you the most productive when you are organized and planning, or are you the most productive when you are "Agile" and reacting to things when they come your way? The Agile proponent will argue, "I don't know what is coming, so I can't plan." What I say is you don't know what is coming because you don't manage your environment and you don't study your environment especially your customers.

It is clear from a practical perspective (and the academic literature supports this also) if you are under a lot of pressure you will not be creative. If you are struggling to keep your head above water and working a lot of overtime you will not be creative or innovative either. The organizational behavior literature tells us teams who plan the most are the most creative, innovative and in turn truly agile. Those teams who plan the least are the ones who are rigid and least able to adapt to change.

The software industry causes their own chaos by their lack of discipline. I will take it a step further and suggest the software industry is collectively dysfunctional. Again the basic premise of Agile Methods is there is nothing I can do about my environment. I am a victim of my environment. I am a victim of circumstances. I can't plan I can only react.

Music

Another common idea in Agile Methods is that adding structure, rules and consistency somehow inhibits creativity, innovation and productivity. Nothing could be further from the truth. Music is a perfect example of this. Music is full of measurement, rules, structure and is consistently written. A quarter note is a quarter note regardless of where it appears in the sheet music. Furthermore a quarter note has universal meaning. Without all of this structure and rules it would be difficult for a musician to perform a piece of music. It would be impossible for an orchestra to come together and play as a symphony.

All structure in music allows the composer to communicate to musicians how the piece should be played. The composer may speak different languages, but they communicate through the music. Music such as *Fur Elise* was written 200 years or so ago by Beethoven whose native language was German. It can be played by my 14 year old daughter who does not speak a word of German. While the content is different the symbols are exactly the same for a contemporary piece like *Linus and Lucy* (the recognized Charlie Brown theme song) written by Vince Guaraldi.

Sheet music is written concisely and consistently. There is nothing on sheet music which is not necessary. A typical piece of music contains measurement and symbols. Unless you are trained to read music you have no idea of the meaning of a symbol like 𝄋

A composer does not decide to develop his own vocabulary or develop their own symbols. Individuals are trained to write music and they are trained to read music. It takes years to learn to be able to sight read music. Within most IT organizations they do not standardize terminology or symbols for their documentation.

Software developers do not like to write documentation because their writing skills are weak. The idea of concise technical writing is not new or unique to music or other disciplines. The book, *Handbook for Technical Writing*, I used over 20 years ago when I was an undergraduate student at Texas A&M University reads,

> *"Concise writing provides exactly what the reader needs and not a bit more or less. Bloated writing with superfluous information serves to confuse and obscure."*

Agile proponents prefer communication between people instead of documentation.[10] They prefer to spend the money on coding instead. Architects communicate with craftsmen using written documentation and they do not rely on verbal communication. Communication without documentation only works in small groups of less than 4 people and it only works during short time durations. It may be okay for a person doing a small home remodel project to rely on verbal communication instead of written blueprints.   If things are not written down, then people tend to forget them.   The number of members of a group dictates how many communication paths exist. If the group has 2 members, then there are two communication paths. If the group has four members, the number of paths increases to 5. If the size of the group is 10, then there are 45 communication paths. If the size of the group is increased to 100, then there are about 5,000 communication paths. I don't think anyone would argue software developers have a reputation of being great communicators either in writing or speaking.

A full orchestra has about 100 members.   A full orchestra communicates using sheet music, and a conductor keeps them on track and together.   There is a tremendous amount of structure and discipline required for a musician playing in an orchestra. Matthew Sigman[i] writes in the Orchestra Manager's Survival Guide the following.

> *"During rehearsals or concerts, musicians experience a total lack of control over their environment. They do not control when the music starts, when the music ends, or how the music goes. They don't even have the authority to leave the stage to attend to personal needs. They are, in essence, rats in a maze, at the whim of the god with the baton."*

There are a lot of similarities between an orchestra and a software development team. Team members spend a tremendous amount of time alone with some sort of instrument and they are required to come together and perform as a whole. A conductor (or project manager) is required to keep everyone on the same page.   Most musicians are introverts; like most software professionals they feel more comfortable with their instrument (in front of a computer) than they do with other people.

Playing the piano is simple, right? My 14 year old daughter plays rather well.   A piano has 88 keys and they are all defined. The sheet music follows rules. Yet the actual playing of the piano is hard. Most people who start piano lessons quit.   I don't think anyone knows the exact percentage of people who start out playing the piano who become virtuosos but it has to be a fraction of a percentage. I would also maintain a

[10] From the Manifesto for Agile Software Development, "The most efficient and effective method of conveying information to an within a development team is face to face communication."

significant number of people who start out playing the piano have bad experiences and most of them quit. The bottom line is that it takes a lot of discipline, practice, and dedication to learn to play the piano. It also takes discipline, practice and dedication to develop software.

Music is written in an iterative fashion, but keep in mind music is written by only one or two individuals. It is not written by a large group of individuals. The final product of a composer is formal sheet music any musician can read and play. Agile proponents prefer playing by ear instead of sight-reading sheet music.

Agile proponents don't get the fact that analysis documentation is just that. It is analysis documentation. It is how you figure out the problem and the solutions. I doubt if any agile proponents would like a construction company to build their homes using Agile methods.

Library Science

Agile proponents do not see the need to document or to organization documentation. This is a common practice of most software development and not limited to just Agile. I ask organizations to compare how they catalog documents to how their local library does it. What would the local library look like if they followed your process for organizing documentation?

How should you organize your documents, so they can be easily found, modified and understood? Do you have any system to organize your documents? Are documents stored in private libraries or are they organized? I have one client who has deployed a type of Dewey decimal system. It does not matter which software application or project is documented; all of them use the same numbering schema.

I searched for the job title of software librarian on Monster.Com. I am not surprised, but none of those jobs required any background in library science. On the other hand librarian jobs in the fields of medicine and law include the following:

You must possess the following requirements and experience to succeed in this position: Master's Degree in Library Science from an ALA accredited program.

What is the motivation for the fields of medicine and law to require a Masters in Library Science to organize their documents while there is no requirement whatsoever for the software industry? The reason is medicine and law organizes their documents is to assist the organization in learning and to be able to easily find information at a later date. Most software development organizations cannot find their application documentation because it has never been cataloged properly.

On any single software project or application, there can be 100's perhaps 1,000's of documents. Most of these documents will reside on personal computers or stored on some shared drive. I have worked with organizations where all "important" documents are stored on a shared drive. The shared drive was created with no standards in mind and no subdirectories. There are no document naming standards either. In other words, there is just a big pile of documents. Within those documents there are no naming conventions, no standard glossary and no standard format. In this situation it can be hard and nearly impossible to find information. The irony in this is that software developers create all kinds of databases and organize information for their clients.

Courtroom

Agile proponents rely on an individual's communication and memory instead of documentation. Everything said in the courtroom is recorded. Extremely detailed documents are created by a trained court reporter. There is a judge, the plaintiff, the defendant, and the jury, so I am sure there is no disagreement on what was said during the trial, right? The jury members are independent observers and you would think nine observers could agree on what was actually said during the trial. Nine independent individual memories are not reliable, so there is a need for a written record of what was said.

The analogy could be utilized regarding gathering requirements and other application documentation. If the room is full of participants and observers, then one would think after the requirements gathering session, the observers could remember everything. If we apply Agile methods to the courtroom there is little need for a court reporter.

Management Science

Another argument of Agile Methods is that professional staff does not need to be managed as the "ole" uneducated factory worker.[11] What about the relationship between hospital professional staff (nurses, doctors, research scientists)? Using the Agile argument, hospitals and healthcare organizations do not need management, planning or process because they are professionals and you can just rely on individuals' professionalism. The same is true for engineering, architectural firms, pharmaceutical, and law firms.

It is a mistake when Agile proponents invoke the name of Fredrick Taylor to support their argument. Using Google I searched the term "Frederick Taylor" Agile and "Fredrick Taylor" Agile. Many Agile websites use the English spelling of Frederick Taylor's name (Fredrick). These searches yielded about 50,000 websites. These websites discuss the idea of Frederick Taylor, industrial revolution management techniques and Agile.

As a member of the Academy of Management (AOM) I don't ever remember reading any article about Frederick Taylor or hearing anyone talk about him. Agilistas argue like there has not been any management research for the past 100 years or so. A few weeks ago, I was at an Academy of Management Conference and I don't recall anyone there talking about Frederick Taylor. The Academy of Management is a professional organization

[11] Several Agile publications encourage "spontaneous self-organization" in lieu of structured project management. From Agile Advice, "No one orders a team or an individual to do specific work." See agileadvice.com.

whose membership includes those teaching and conducting research in the field of management. The AOM has over 16,000 members and hosts conferences with nearly 8,000 attendees. The AOM has 5 different monthly journals. In fact, there are over 80 academic journals dedicated to organizational behavior and management. There has been a lot of management research since Frederick Taylor and most of it has nothing to do with factory workers.[ii] The far majority of management research today deals with information workers, service workers and professional staff.

The real legacy of Taylorism is his emphasis on breaking down everything into small steps, and his prediction of "a choreography" of work leading to previously unimaginable productivity gains. One of Taylor's most controversial ideas is labor and analysis is divided. The boss plans and the hired man executes. With knowledge workers the role of planning is not necessarily the boss but a member of the team. The role is divided because the skill set to plan is a specialty just like writing code. Planning is centralized to prevent duplicate efforts.

War Room

Parts of Agile methods follow a war room approach. The most practical application of war room is the ER (emergency room). While the ER is a great method to stabilize patients it is not an effective way to treat them. The primary objective of an emergency room is to stabilize patients.

A war room is an effective method of handling a crisis, but not an effective method of managing day-to-day operations or designing a software application. While the ER serves a very specific purpose, not every patient is treated using ER techniques, nor does every patient at the hospital pass though the Emergency Room. Once a patient is stabilized they are moved to the care unit that can best treat them.

The Agile method encourages self-organizing teams without established processes. In an ER there is only one person in charge. Just like the conductor of an orchestra the head ER physician and head nurse call all the shots. There is little discussion and few questions. The ER staff relies on past training and established processes to handle most situations.

Calatrava

Agile proponents believe documentation is an overhead cost and should be reduced or eliminated. They believe documentation should be done on 3 x 5 index cards. The money spent on documentation costs would be better spent on coding.[12] I am not sure what is supposed to be done with the 3 x 5 index cards. Maybe the programmer is supposed to keep them in his shirt pocket behind his pocket protector. In my opinion documenting on a 3 x 5 index cards is one evolutionary step beyond writing code on the back of a napkin.

[12] Larman, Craig. Agile and Iterative Development: A Manager's Guide, The agile software development series. Boston: Addison-Wesley, 2004. See 153.

These days I have become a bit obsessed with Santiago Calatrava. He is an architect from Spain who designs buildings all over the world. It is important to understand he is not just the architect, but also the project manager. His structures are considered some of the most creative structures ever built by a human. A common criticism of comparing software with the construction industry, or for that matter every other industry, is that every software project is unique and different from any done before.

Calatrava does not document by writing specifications on 3 x 5 index cards. Calatrava does not make up symbols when he constructs a blueprint nor does he mix symbols. He uses architectural symbols that can be understood by any member of his design team as well as any member of the construction team.

Calatrava builds a variety of custom buildings. He builds Olympic stadiums, museums, bridges, shopping malls, train stations, office buildings and even memorials. He designs buildings in his office in Switzerland that will be built in cities like Toronto and Milwaukee. He designed the Milwaukee Art Museum and the award winning BCE Place in Toronto Each of these types of buildings has different function, but there are similarities. The blueprint documents utilize the same symbols for windows, doors, and entry ways, which improves communication between all the parties.

There are a lot of similarities in the software development process. There are onlines, inbound and outbound interfaces, reports (some on-line and some on paper) and files for storage. That is just about it. When gathering requirements a developer should bring current application documentation with them to help customers understand their own needs and where changes can be made.

Can you imagine what extreme construction or Agile construction methods would look like? In fact, would you turn over 200-300 thousand dollars to have your home constructed using the same principles advocated by Agile or Xtreme programming? With extreme building you just get some wood and bricks, then start building. Helping the customer figure out what they want would not be part of the extreme building process.

Mathematical Proof

There is a branch of mathematics called set theory which actually disproves Agile[13] (or iterative) development. If you do not like math you may want to skip this section.

A software application is in essence a defined set of requirements. When these requirements are combined they interact to form what we call a software system or software application. A requirement is not totally independent existing alone. Rather, requirements are woven together, becoming interdependent. It is common to have requirements intersect.

[13] Dadkhah, Kamran. Foundations of Mathematical & Computational Economics. Mason, OH: Thomson South-Western, 2007.

Imagine you only have two requirements A and B. If these two requirements are independent then they can be built separately without any consideration for each other. What happens if these two requirements are dependent and intersect? If requirement B intersects with (or is dependent on) A, then when B is built A has to be modified. Using set theory it becomes self evident that:

$A + B + (A \cap B) > A + B$

Now if a new requirement C is discovered and both A and B are dependent on C, then when C is built both A and B have to be modified. Again it is self evident that:

$A + B + (A \cap B) + C + (C \cap A) + (C \cap B) > A + B + C$

The problem of course is iteration and "reworking" what has already been done before. It is a normal part of the Agile method to keep reworking the code that was written before. Agile has time to do things over and over again (iterate), but they do not have time to define things upfront. This is similar to watching an ant farm. It only appears that everyone is busy and productive. Look, everyone is coding! Look, everyone is productive!

Sometimes it is hard to get a read on what is Agile because they misuse or misunderstand terms like empirical process, empirical theory, empirical model and queuing theory. Queuing theory has nothing to do with breaking down a process into small parts.[14] Queuing theory was developed to help managers understand and make better decisions concerning the operation of waiting lines.[15] As any person from Great Britain will tell you a waiting line is known as a queue.

Queuing theory is based upon the idea of first-come, first-served. Queuing theory proved a single channel queue is more effective than a multi-channel queue. In multi-channel queue there are multiple lines and customers choose the line. We see this at grocery stores. In a single channel environment there is only one line and customers queue up in a single line. Most airlines and some retail outlets have adopted a single channel strategy. The reason single channel is more efficient and waiting times are less in single channel queues are less than multi-channel queues is because customers are not good at choosing lines. The same is true with buffets. Buffets move much efficient when customers queue up and are served instead of serving themselves.[16]

I would think queuing theory would contradict Agile because coders are picking the work instead of it being "served" to them by a project manager. Nonetheless, this is another example where Agile is either misunderstanding terms or misleading.

[14] http://www.stsc.hill.af.mil/Crosstalk/2002/10/bowers.html

[15] Anderson, David Ray, Dennis J. Sweeney, and Thomas Arthur Williams. Quantitative Methods for Business. St. Paul: West Pub. Co, 2004.

[16] http://www.bbqcatering.com/sub/queuing_theory.pdf Queuing Theory and Practice for Caterers.

Please Step Away from the Code

The greatest improvement in the productive powers of labor, and the greater part of the skill, dexterity, and judgment with which it is anywhere directed, or applied, seem to have been the effects of the division of labor
Adam Smith, Wealth of Nations, 1776.

Agile is squarely focused on the code.[17] I was walking in a parking lot and walked too close to an automobile. I heard the words from under the hood, "please step away from the car." I was a bit surprised, so I took a few steps back. I thought about this and I stepped close to the car. Again the words from under the hood, "please step away from the car." I wish there was a voice that could radiate from software projects that would say, "Please step away from the code." More software developers need to step back away from the code, lift their heads and look around at their customer. They need to study their environments, their industry, and their customers. They need to stop focusing on the code.

By focusing on the code, Agilistas move the software professional towards becoming a craftsman, not an engineer or designer. Programmers are the craftsmen of the software development industry. An engineer is concerned with creating specifications, designs and plans. A craftsman tries to create a product to meet the specifications and designs created by the engineer. An electrical engineer may design a power grid but the craftsman (electrician) actually goes and hangs the powerlines. A mechanical engineer designs the car and the automaker builds it.

As industries matured the role of craftsman and engineer separated and there was specialization. Agile encourages no specialization whatsoever. This is rather unique especially when we examine the idea of specialization in other industries and disciplines. The best comparison is with medicine and how medicine specialized during the nineteenth century. Medical historians argue the rapid expansion of knowledge in the field of medicine did not cause specialization, but it was the desire to move the industry forward that caused specialization. Jean Emmanuel Gilibert put it more succinctly, "specialization was necessary for the advancement of medicine."[18] Today many in the field of medicine maintain that without specialization, medicine would not have moved forward as rapidly as it did. If we want to move the software industry forward, then there needs to be specialization.

Specialization preceded the idea of waterfall methods. It was specialization that leads to the waterfall development model not the other way around. It was not a desire to have waterfall that lead to specialization. A specialist is more productive than a generalist. In

[17] Martin, Robert C. Agile Software Development Principles, Patterns, and Practices. Alan Apt series. Upper Saddle River, N.J.: Prentice Hall, 2003. The book has 500 pages. Of those 500 pages 400 or so are dedicated to coding.

[18] Jean-Emmanuel Gilibert, *L'anarchie médicinale, ou La médecine considérée comme nuisible à la société*, 2d ed., 3 vols. (1772; Paris: n.p., 1776), 3: 221. On Gilibert see Samuel Kotteck, "'Citizens! Do You Want Children's Doctors?' An Early Vindication of 'Paediatric' Specialists," *Med. Hist.*, 1991, *35:* 103–16.

the early nineteenth century the seamstress performed all the roles necessary to make a dress. As the industry matured specialization began to happen. The seamstress became a button sewer. The design role became a specialization also. All the roles once performed by the seamstress were performed by several specialists. This caused the waterfall development method.

A final thought

Before I conclude, I want to mention I have been personalyl attacked regarding this article and my ideas on Agile. The extreme programming yahoo group invited me to participate in their group for what they called an "intelligent debate." It became a forum for Agilstas to name call and sling insulting comments. As the moderator of the yahoo group wrote, "I'm not sure what started this particular round of name calling, but lets stop it. NOW"

The term *Delenda est Longstreet* was used against me. The terms *Agilista*, *Manifesto* and especially *Delenda est* are associated with violence. The term Agilista is a derivative of Sandinista. The Sandinistas led a Marxist revolution in Nicaragua killing thousands of innocent people in its wake. Manifesto is most commonly associated with the Communist Manifesto and the Unabomber had a manifesto too. The expression of *Delenda est Carthage* meant the total and complete destruction of ancient Carthage. It is the first recorded incident of genocide. A logical and reasonable person has to ask the purpose of using such aggressive, violent and offensive language.

Unfortunately Agilistas do not stop at just insulting language. I have received emails from individuals telling me how Agilistas have insulted them in their own companies. After one of my presentations a woman told me, "I disagreed with some Agile consultants at my company and the next thing I knew they were trying to get me fired." Disagreeing with Agilistas generally leads to a series of personal attacks and insults.

Now In Conclusion

Those adopting Agile methods only think they are "winning" and improving because they don't know. By making the argument that software development is not predictable or measurable, they will never have empirical results and they will never learn. Without careful study and examination you can't determine if performance has improved or not.

The Agile proponent believes the software development environment is not predictable and not measurable. Most Agile proponents only have anecdotal information not empirical data. They are violating their own argument of an empirical process, model or theory.

History tells us as things change there are those who refuse to progress and move forward and hold on to the past. The legacy of software development is a legacy of an undisciplined wandering industry. Few would argue the legacy of software development is one of discipline and structure. In actuality, Agile proponents are those holding onto the past. They want to formalize the idea of being undisciplined and unstructured.

The whole point of this article is software development inflicts most of their own wounds. Agile is just formalizing the process of self-inflicting wounds. There are solutions to software development problems other than jumping off the Agile cliff. Agile is not necessarily unique from software development in general, they just want to formalize bad practices.

Agile proponents believe software development is unique from all other industries. They believe there is nothing to be learned from other disciplines like music, library science, medicine or architecture.   Like most individuals in a crunch they default back to a position of comfort. For software developers this is coding. It is not about the code it is about creating products that customers can use.

Agile proponents argue against structured methods because it enforces accountability. Too many software developers want to be left alone so they can code.   They want to run free and not be under any rule.   They avoid accountability, management, or the idea of reining in costs.

Agile proponents discourage documentation. The software industry does not have reputation of being great communicators or organizers.   Instead of trying to improve communication skills, they abandon the idea of communication. Instead of organizing documents they prefer to just not document. It is not possible for large groups of individuals to communicate without structured documentation.

Agile proponents discourage a division of labor and argue all software roles are interchangeable.

Agile proponents are the leaders of a dysfunctional industry. They honestly believe no other industry or person can provide good requirements. They also believe nothing can be learned from other industries. They feel, “It is not the software industry that needs to change, but everyone else”.

---

i Matthew Sigman is a conductor, author and senior member of the American Orchestra League.

ii From Martin Fowlers website, “Deciding that people come first is a big decision, one that requires a lot of determination to push through. The notion of people as resources is deeply ingrained in business thinking, its roots going back to the impact of Frederick Taylor's Scientific Management approach.   In running a factory, this Taylorist approach may make sense. But for the highly creative and professional work, which I believe software development to be, this does not hold. (And in fact modern manufacturing is also moving away from the Taylorist model.)”