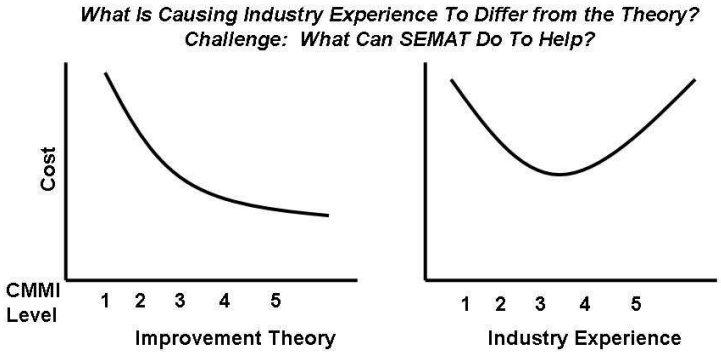
What Should Software Engineering Consist Of?
An Industry Experience Perspective

by Paul E. McMahon, PEM Systems, pemcmahon@acm.org

This paper presents a proposed approach addressing the goals of SEMAT[1] to: "*re-found software engineering based on a solid theory, proven principles and best practices.*" The proposed approach is presented as a starting point for discussion and is based on experiences over the past 15 years helping high maturity software organizations[2] increase their Agility and helping Agile growing organizations increase their process maturity using the CMM/CMMI model. A practical position on what it means to "*engineer software*" is included. To help motivate the discussion the following diagram and challenge is presented based on observations in industry today.



I will start with a description of a proposed SEMAT kernel containing four core[3] elements including a Technical Data Package (TDP) Element, a Stakeholder Element, a Communication Element, and a Measurement Element, along with rationale showing how the core elements provide the desired SEMAT kernel features[4]. Following this description I will discuss what else is needed to address the SEMAT identified problems based on industry observations today.

Technical Data Package (TDP) Element

The TDP Element includes five sub-elements: *Requirements Artifact, Design/Architecture Artifact, Software Artifact, Verification Artifact, Usage/Support Documentation Artifact.* The five sub-element artifacts are based on critical product and skill needs observed in multiple high maturity software organizations. Today what we are teaching in the Universities to prepare our next generation of software engineers lacks the needed balance across these five sub-elements with respect to theory, methods and proven "*how to*" best practices. I will explain more later what I mean by "*how to*" best practices, but let it suffice here to say software practitioners today need more help connecting theory to practice. A *concrete product-centric* kernel, as proposed, can help bridge this current divide and provide a sound basis to re-found software engineering.

Stakeholder Element

The Stakeholder Element includes three sub-elements: *Developer, Reviewer, and Approver*[5]. Inadequate stakeholder involvement is a common root cause of immature Software Engineering in industry today. Under the common cost and schedule pressures today in industry too often software practitioners fail to involve the right people at the right time. The *Developer, Reviewer, and Approver* model is based on what I have observed working today in mature software organizations.

---

[1] http://www.semat.org/bin/view
[2] Mostly organizations serving the US Department of Defense Community.
[3] The term "core" used throughout this paper means a minimum set that can be extended, but not reduced.
[4] http://sematblog.wordpress.com/2009/12/18/what-is-the-scope-of-the-kernel/
[5] The customer is included in the "Approver" sub-element.

Communication Element
The Communication Element includes the following sub-elements: *Task and task responsibility identification, progress/obstacle tracking/reporting, and a Decision-Aid Framework.* I am not here proposing "best communication practices", such as daily standup meetings common today in many Agile organizations. Rather I am proposing to address the *essentials* of what needs to be communicated. Today, in practice, often due to project pressures inadequate communication occurs in terms of *task status, current decisions faced, options and consequences.* Software practitioners need more "*how-to engineer software*" guidance. This is not meant to imply "*how-to*" in terms of methods, tools or specific practices, but rather "*how-to reason*" to find the right answer given specific project conditions at any point in time. While software practitioners do need to ultimately understand "how to" in terms of methods, tools, and specific practices, where we are most critically failing today is in teaching them "*how to*" evaluate and decide on the appropriate use of methods, tools and specific practices given specific project conditions. This is a key missing ingredient in Software Engineering today that will be discussed further shortly.

Measurement Element
The Measurement Element includes essential measurements related to the *TDP, Stakeholder, and Communication Core Elements.* We must measure the implementation to manage it.

The four core kernel elements being proposed are based on essentials drawn from experiences using the CMMI model, and what is working today as observed in multiple mature software organizations. The proposed kernel is *practice, language, method, and life cycle independent.* It does not specify *how-to* details. The kernel is product-centric making it *concrete*. It is *scalable* and *extensible* as it is based on practices observed in mature software organizations ranging from extremely small (less than 20 people) to Mega-Organizations with thousands of people. ***A next step for a SEMAT working group could be to define the core for the sub-elements.***

What Else Is Needed to Address the SEMAT Identified Problems?
The proposed *artifact-centric kernel* previously discussed doesn't fully bridge the current theory-practice Software Engineering divide observed today in industry. Starting with a concise concrete kernel independent of life cycle, language, methods and practices helps in developing the kernel by avoiding the potentially never-ending battles of differing viewpoints on methods and practices. Nevertheless, this approach doesn't help software practitioners and leaders when it comes to *how-to* make good *software engineering decisions.*

Alistair Cockburn has made the point that when we think of "*doing engineering*" in areas such as Civil Engineering we think of looking up previous solutions in "code books". However, since change occurs rapidly in the software arena we have few "code books" with previous answers that we can rely on today. As a result, Alistair suggests a better way to think about "engineering software" is in terms of making the "*tradeoffs*" and "*decisions*".[6] I support this view of Software Engineering and I believe this presents the SEMAT community with an opportunity to provide value-added assistance to software practitioners and leaders today and it can help to align academia, research and industry on what it means to "*engineer software*." Through the proposed kernel we begin to concisely and concretely define what "engineering software" means by taking a practical product-centric view. However, we must go further to provide assistance helping software practitioners and leaders make more effective "***tradeoffs***" and **"*decisions*"**.

Patterns For Success To Aid Software Engineering Tradeoffs and Decisions
What is needed today in practice is better guidance in "*how-to*" make those decisions related to scaling, methods, life cycle, best practices, tools, and training-- not to give the answers--but to help with the ***reasoning process*** one must go through to make the best choice given specific

[6] Agile Software Development: The Cooperative Game, 2nd Edition, Alistair Cockburn

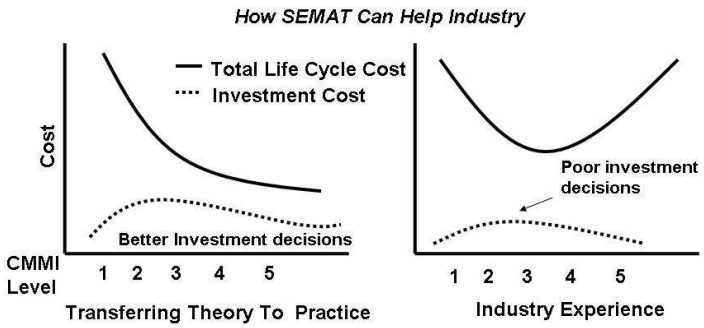
project constraints and circumstances. This leads to a question: ***Where can we find this "how-to" guidance so it can be institutionalized as part of "software engineering"?***

While it is true that today software engineering continues to be hampered by "immature practices," at the same time there exist many success stories of software organizations that have reached high levels of maturity producing quality software rapidly despite facing continual changing requirements and technology. Many of these successful organizations have found ways to increase their productivity and competitiveness by automating parts of their software development and through domain-specific solutions and tools which support more effective *decisions* by simplifying their most commonly occurring decisions.[7]

I suggest that while the "*reasoning process used by these organizations to achieve high software maturity and business success"* may not yet have been captured in a systematic way so that it could be trained and institutionalized, that much of this knowledge is known today and could be captured, trained and institutionalized. I also suggest that there exist *common patterns* to what many of these successful high software maturity organizations do today and that there is an opportunity for the SEMAT working groups to capture this information so that it could be treated more systematically as part of what it means to "*engineer software*." Potentially this effort could lead to improvements in Software Engineering Curriculum in Universities to better prepare today's students for the challenges they will face when entering industry.

Today when I observe immature software practices in organizations often the cause is found to be a lack of understanding of the *questions that should be asked, when they should be asked, who they should be asked to, and how to make related decisions that consider the right factors and potential consequences.* Often this situation is found to tie back to poor organizational investment decisions. One potential valuable output of SEMAT could be a ***decision-aid framework*** to help both software practitioners and organizational leaders make better decisions based on sound software engineering practices and common lessons.[8]

Software investment decisions today typically include tools and training frequently focused on technical skills often missing critical *human interaction* and *decision-making skills*. Today conflicting views on how to translate software engineering theory to practice leads to poor decisions that are not based on *measureable value to the customer* leading to a lack of promised investment payback. A product-centric measurable kernel can help, but industry also needs help with the "*how-to*" associated with *critical related decisions*. Refer to diagram below.



I believe it was Barry Boehm who once said "we can't all be Kent Beck." Part of the path out of today's common immature practices is to provide help to the software practitioners and leaders in the trenches with the common decisions faced each day on today's software projects.

---

[7] http://www.link.com/pdfs/iitsec-4-2002.pdf, http://www.stsc.hill.af.mil/crosstalk/1995/03/Pattern.asp

[8] http://www.stsc.hill.af.mil/CrossTalk/2008/05/0805McMahon.html, http://www.stsc.hill.af.mil/crosstalk/2006/05/0605McMahon.html