# Read My Lips: No New Models![1]

**Karl E. Wiegers**

Process Impact
www.processimpact.com

Recently, I had dinner with James Bach, a thoughtful and well-known figure in the software quality industry. At one point, James began to tell me about a new testing model he was developing. I gently interrupted him and offered the opinion that, with a few exceptions, the software industry does not need any more models right now. We don't need a lot of new formalisms for software development, new design methods, new life cycle approaches, new frameworks for process improvement, or new quality models.

What we *do* need is for practitioners to routinely and effectively apply the techniques defined by our existing models and frameworks. Once we've reached the practical limit of these approaches, we can turn to improved models that provide guidance for working in better ways. To be sure, some current approaches may be unworkable, and projects on the bleeding edge of technology, business needs, or development approaches may find current methods inadequate. More sophisticated models may also benefit practitioners who have already pushed current methods to their limits. My sampling of audiences at conferences and training seminars suggests, though, that many organizations are not consistently applying existing approaches for software development excellence.

I suggest here several sets of software engineering and management practices that, in my experience, are still not being routinely applied across the industry. The problem is not always a lack of suitable models to help us structure our thinking in these domains. The real problems often include:

- insufficient awareness of current best practices and published standards in software development, management, and quality;
- inadequate training of practitioners and managers in these established practices;
- resistance to change, as evidenced by Not Invented Here syndrome and an insistence that "our project is different and those things don't apply"; and
- a shortage of discipline, rigor, and available time for individuals to continuously improve their personal software processes by applying a broad spectrum of superior techniques.

[1] This paper was originally published in *IEEE Software*, September/October 1998. © 1998 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

## Why Create New Models?

New approaches to software development and management are promulgated for a variety of reasons. Some organizations have pushed the limits of current approaches and really need a better way. Early adopters like to create and try out new approaches, thereby serving as a valuable test bed for the rest of the industry. Sometimes it's a matter of packaging old wine (e.g., scenarios) in new bottles (e.g., use cases). Often, frustration with the inability to apply a cumbersome current model in a sensible and practical way leads to the next generation of models in some domain. Other times, new approaches may be developed simply because the ones already available are not being adopted by the practitioner community. If a novel context helps a sound approach find more common acceptance and push our profession forward, that's progress.

Unfortunately, we sometimes find ourselves trying to sell a better mousetrap to people who don't even realize they have mice. As an example, some 315 software engineering standards are currently available, of which those from the IEEE are perhaps the most widely known in North America.[1] However, my informal conference samplings suggest that less than 10% of the audience members have access to the *IEEE Software Engineering Standards Collection*. It is difficult to conclude the standards are not useful, if you either aren't aware of their existence or haven't tried to follow them.

## Process Improvement

A plethora of models have been developed to guide software process improvement programs and process assessments, including the Software Capability Maturity Model (CMM), Trillium, SPICE, Bootstrap, TickIT, and others. Recently, new models have been advanced to address personal and team software processes. Yet, many organizations do not have successful process improvement programs in place today. Many practitioners either have not heard of the CMM, or they hold large misconceptions about it. The Software Engineering Institute (SEI) has engaged in a multi-year effort to revise the CMM, but I don't think this effort will result in more organizations applying it effectively.

Managers and change agents should examine their current process improvement efforts. Are your teams applying *any* model effectively? Are they being sensible, not dogmatic, about implementing improvements? Are they fixing current problems and addressing risk areas, rather than chasing maturity levels and certifications? And, ultimately, are practitioners working in new, better ways (the bottom line in process improvement)?

If the answers to these questions are "no," the solution is not to concoct yet another process improvement model. Instead, those of us in the process improvement business need to educate our clients and cohorts on the intent and application of tools like the CMM. Using an established framework to guide your process assessment and improvement activities is highly valuable. But almost any of the existing choices will do.

We need to be flexible and nondogmatic in our interpretation of these models. Use them as guidebooks of structured wisdom that can help us achieve our objectives of improved business and technical success through process improvement. Conforming to the model's expectations should not become an end unto itself. If you try all these approaches and still find that none of the existing improvement models fits your needs, perhaps then we should search for new and innovative approaches.

## Testing

A software quality magazine recently published a series of columns delineating an elaborate model for system testing. Several “testing maturity models” have been proposed by leaders in the software quality industry. While some very large, very regimented projects may actually apply these complex testing models, I think the real testing issues are closer to home.

Think about the developers in your organization who also do some testing. Do they have testing books on their desks? (My informal surveys of conference attendees suggests not.) Have they been trained in testing? (Ditto.) Do they write test plans? Are their tests documented and repeatable? Do they understand basic testing concepts? Can they describe the state of a program after testing is complete?

If practitioners cannot answer such questions in the affirmative, a better testing model probably won’t help them. Some training on testing practices and concepts, test case design and documentation, and the use of automated testing tools will do more good than a new testing model. Experienced testers can enhance their effectiveness by following the dictates of a rigorous testing model. However, formulating ever more elaborate models of the testing process will do little to improve the way average software developers test their products.

## Design Methods

A multitude of design methods have been hawked over the past decades. First we went through the structured methods revolution, which brought us useful techniques such as data flow modeling, entity-relationship diagrams, state-transition diagrams, Warnier-Orr diagrams, and others. This was followed by a series of object-oriented approaches, including CRC cards, Object-Modeling Technique, Unified Modeling Language, and various other methods named after various methodologists. Learning to model software systems through application of structured techniques was a turning point in my software development career, as I began to depict, understand, and improve systems in a disciplined way before constructing them. I certainly do not suggest that analysis and design modeling is anything less than a key element of solid systems development.

But how does your team really do design today? Do team members spend much effort at all between requirements and code? Are their designs diagrammed in any formal way? Has the group standardized on the design notations to use? Do they use automated tools to draw design models? Are the designs improved through iteration? If not, will the next design methodology to come along get them to contemplate design before cutting code?

A lack of suitable methods and modeling techniques is probably not the bottleneck to getting more development teams to perform rigorous design. More likely, it is the ability to understand any design approach well enough to make it useful, the willingness to invest in necessary training and tools, and an appreciation that improving design solutions through iteration is an effective route to higher quality, more robust systems.

## Inspection

An industry guru once told me that formal inspections of software work products are being practiced by about 50% of software groups; several of my fellow consultants agree that 20% is a more likely figure. Many practitioners do not understand the differences between informal design reviews or code walkthroughs, and formal inspections. Reviews to communicate information are not differentiated from those intended to find defects. Three major books on inspections describe similar approaches, none in less than 350 pages. Purists debate the effectiveness of various inspection steps and methods, and new inspection methods are touted as superior with little empirical evidence.

But does your team routinely inspect software work products of all types? Are they trained in inspections and reviews? Do your inspections actually find defects? Do you collect and use inspection metrics? Are inspections part of your software engineering culture? If not, is it because you're waiting for a better inspection method to come along?

Despite decades of strongly positive experience and recognition as an industry best practice, formal inspection remains an enigma to many software professionals.[2] In most cases, new models for inspection are not required. Instead, let's start with education about the process and benefits of inspections, and some guidance about how and when to use them. Pragmatic, readable books that help developers apply effective inspections as part of their standard software practice will go farther than debates over nuances of approach in one inspection variant or another.

## Risk Management

Risk management is becoming appreciated as a major component of effective software project management and an industry best practice.[2] The SEI has developed elaborate approaches for multi-day project risk assessments, a detailed taxonomy of project risks, models for Software Risk Evaluation, Continuous Risk Management, and Team Risk Management, and a Risk Management Map.[3,4]

But how do your projects perform risk management today? Do your project plans contain even a simple list of risks? If yes, have they been analyzed for probability and impact? If yes, do you have plans for mitigating the most severe? If yes, do you execute and track progress of those plans? If yes, are those plans effective? If yes, do you record the risk management lessons learned in a database for the benefit of future projects? If the answers to these questions are "no," how badly do you need a different risk management model?

My samplings suggest that many projects do not yet practice systematic risk management. Complex models that define comprehensive approaches to risk management will not address this shortcoming, although they can help those who already perform risk management reach the next level of sophistication. More projects, though, need to begin simply documenting their major risks and mitigation strategies, taking positive actions to control them, and tracking progress on risk control. Risk management should become a routine topic of discussion at software development conferences, not just those dealing with process improvement and project management.

## Metrics

Several metrics leaders have suggested "dashboards" or "control panels" of key indicators that software organizations and projects should use to track status. We're encouraged to develop "balanced scorecards" to monitor our projects and organizations, and yet religious wars are waged over precisely what to measure. Literally hundreds of aspects of software products, processes, and projects can be measured, and somewhere there's probably a model that includes each of them.

But what's the state of software measurement where you work? Does your organization measure anything about its projects, products, and processes? Are you collecting reliable metrics in multiple areas? Do you use the data to understand, to take corrective actions, and to predict? Does your team have a healthy "measurement culture"? If not, will a better metrics model really get you started?

Elaborate measurement models are of little help to groups that currently collect little data about their projects, lack a basic understanding of software metrics, or have a culture that avoids measurement because of a fear of data misuse. Such organizations need basic education in software measurement, simple tools to help them get started, and clear ties between their metrics and their business objectives. Less obtrusive ways to collect software measurements that are better integrated with the development process could reduce some of the barriers.

Certainly, there are deficiencies in current metrics, such as those dealing with the sizes of software products, and collecting software metrics is often tedious. But I do not believe a lack of adequate measurement models is the limiting factor that prevents organizations from beginning to quantify certain aspects of the software work they do.

## Practice What We Preach

I don't think models are bad. I have found models that help structure my thinking and provide a framework for making sensible decisions to be extremely valuable. My point is that the software industry is not fully exploiting the models, standards, and frameworks we already have available. Before we invent new models, let's help developers, managers, and quality professionals consistently and effectively apply the practices embodied in those that currently exist.

As educators, let's incorporate a solid foundation of software industry best practices, along with guidance about how to put them into action, into our curricula. As managers, let's emphasize continuous learning in our organizations and reward those who apply better ways of working. As practitioners, let's read the literature and commit to improving our personal software processes through effective application of what we've learned from others. And as industry leaders, let's not clutter the market with Yet Another Model until we're convinced the ones we have are truly not getting the job done.

## Acknowledgments

I am grateful for the insightful comments made by reviewers James Bach, Johanna Rothman, Wolfgang Strigel, and Doris Sturzenberger.

## References

1. S. Magee and L.L. Tripp, *Guide to Software Engineering Standards and Specifications*, Artech House, Boston, Mass., 1997.

2. N. Brown, "Industrial-Strength Management Strategies," *IEEE Software*, Vol. 13, No. 4, July 1996, pp. 94-103.

3. R.P. Higuera and Y.Y. Haimes, "Software Risk Management," Technical Report CMU/SEI-96-TR-012, Software Engineering Insitute, 1996.

4. E.M. Hall, *Managing Risk: Methods for Software Systems Development*, Addison-Wesley, Reading, Mass., 1998.