THE GEORGE WASHINGTON UNIVERSITY
Faculty of Engineering Management & Systems Engineering
Washington, D.C. 20052

21st Century Software Engineering

By Michael Stankosky, D.Sc. [mstanko@gwu.edu]

Introduction

Issues of security, reliability, complexity, cost, and relevancy have always plagued software. We see it everyday by the numerous security breeches and hastily created patches. We see it everyday by having to take it all (bundling) or else nothing. We see it by the continuous promises of greater stuff, and once again, delays after delays (witness VISTA). We see it in the perplexed looks of those who get the new product, and have to hire a PhD geek to help them use it, let alone service it, when it fails; and fail it certainly and mysteriously will. Even U.S. government-mandated software capability maturity model certification, on the part of its software developers, has not ensured error-free, relevant software. Show me a government software development project on time, in budget, and 100 percent specification-proof.

But, that was the 20th century software paradigm that we were subjected to. Meet the 21st century paradigm: open source; global collaboration by customers and software developers; bug-free and secure code; software-on-demand; enterprise management engineering.

New Perspectives

- If it isn't broke, break it.
- Knowledge is the currency of the day
- Collaboration is the wallet of the day.
- The world is flat.
- The Internet/Web is the "21st Century" computer/desk-top/server/knowledge & information conduits.
- 3Cs: collaboration, convergence, codification.

Formula

GoodSoftware = C3squared

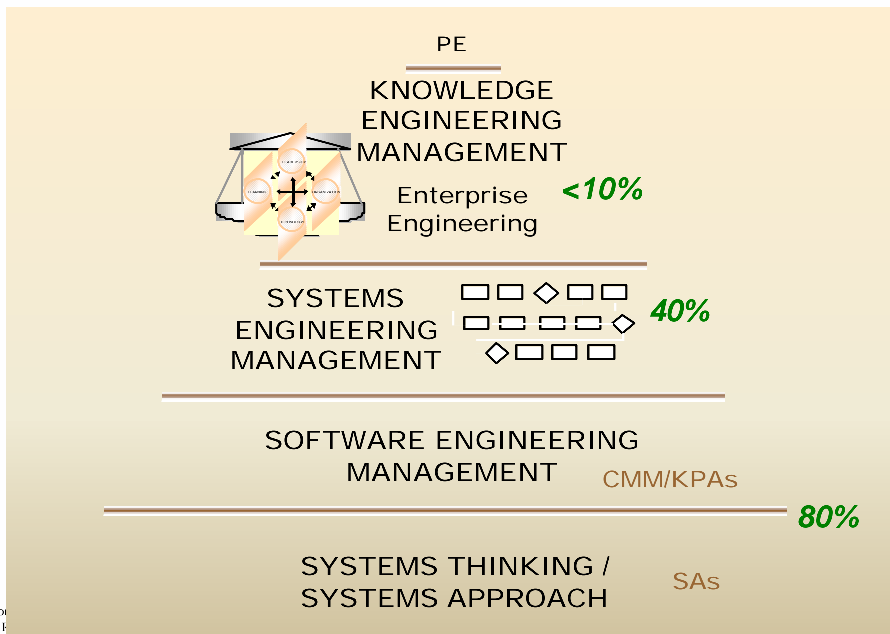
Stankosky's Software Engineering Laws

- It's not about the software.
- Software @ the speed of business.
- If you don't want bugs in software, don't put them there in the first place.
- Collaborate – Collaborate – Collaborate!
- Let the user be the designer – not the software guru.
- Less is more. Even lesser, is most.
- If at first you don't succeed, get someone else.
- Design twice; test four times, code once.
- Software supports operations; not the other way around.
- Design with integration and interfaces in mind.
- Design with the enterprise in mind.
- Design with globalization in mind.

Case Study

The figure below has a percentage next to each engineering discipline (i.e., 80%, 40% and <10%), which represent, based on author's observations and conjecture, failure rates of large, complex software programs, using best practices: Capability Maturity Models (CMMs). Notice that when we superimpose the Systems Engineering CMM on the Software CMM that the failure rate is halved, and finally, when we impose the Enterprise Engineering model, it is less than 10%. It stands to some logic, since software supports a system, and a system supports the enterprise. It is important to start from the top down in software engineering. However, before we had a CMM for Enterprise and Systems Engineering respectively, we did software development first, without the benefit of truly knowing what systems and enterprise the software had to fit into Yes, in a sense we knew, but we did not have the rigor of systems engineering and eventually, enterprise management engineering, to guide us when we were doing our software engineering. Imagine the financial and program consequences if such an approach worked, and were fully implemented. Unfortunately, the newspapers recently reminded us of the large system development failures by the largest corporations, despite their using the best practices in software engineering.

Are Your Sure About the 20% Solution?



©Copyright by Michael Stankosky, D.Sc.