

Report on the Texas Symposium on Software Engineering

August 27-28, 2004 Austin, Texas

(provided by Scott Duncan)

This was the first TSSE and was sponsored largely by the University of Texas at Austin's Computer Science and Lifelong Engineering programs. The conference chair was Herb Krasner who started the Software Quality Institute at UT (<http://sqi.utexas.edu>). In his opening remarks, he stated that TSSE will focus on software system successes and failures and the associated lessons learned. TSSE is intended to cover "advanced topics of interest to the Texas-based software development community." He hoped TSSE could help guide UT as to what they should be doing in their Software Engineering program and to encourage the software community in Texas to examine how they approach designing and implementing systems. "Software Engineering has gotten a bad reputation," he said, due to the lack of professionalism and background in engineering rigor of many who develop software.

My involvement with TSSE was based on my IEEE work on agile methods and my association with Krasner during my time in the Bell System. I was one of the contacts between Bell Communications Research and the Austin-based Microelectronics and Computer Consortium (MCC) where Krasner conducted one of the industry research programs in software design during the early 90's. He invited me to come and present a tutorial on agile methods and participate on a "methods" panel. Friday was all tutorials, so the actual Symposium was on Saturday. What follows are notes from the Saturday sessions.

Keynote: Risks in Software Development (Peter Neumann)

Peter Neumann is perhaps most well known for his *Communications of the ACM* column and Internet newsgroup (comp.risks) on (software) risks. He is now at SRI (Stanford Research Institute), but he was also the manager at Bell Labs for whom the originator of Unix™ (Ken Thompson) worked when Unix™ began. Over the years, he has collected thousands of reports on software risks and system failures showing a "long history in a lack of understanding of what the problems are."

"Too many systems," he said, "critically require 24/7 monitoring and response," placing great reliance on human vigilance (i.e., systems administrators). We also need to be able to "predictably satisfy critical requirements" since one of the major sources of weakness in systems includes erroneous and/or incomplete requirements. Others weaknesses include: hardware/software defects, poor human interfaces, malicious/accidental misuse, privacy violations, environmental quirks, "seriously deficient education/training with regard to systems" and the over-endowed infallibility attributed to computer systems,

Regarding the latter, Neumann said, "Believing in system infallibility seems foolish." Yet it is assumed all the time. How often are the data in (and behaviors of) computer systems assumed to be correct? On the other hand, does a typical user have any other data with which to question a system? Neumann said that being trustworthy means what it says, i.e., being worthy of trust. Trust, then, is a belief in a thing's trustworthiness. What is it that makes something "worthy"? What evidence do we demand? Neumann said that trust was closely related to dependability, but, in many common situations, a person simply has no other choice but to accept (not really "trust") the results from a computer system (or doesn't know better). [Refer to DeMarco and Lister's book, *Waltzing With Bears*, for a discussion of risk and one's "right to believe" in something.]

Among the examples Neumann offered of systems with obvious risks not addressed or for which no mitigation strategies exist are:

- ? Plans for automated vehicles that could only be repaired by approved dealers and whose manufacturers could modify vehicle functions through radio broadcasted software downloads.
- ? Electronic voting systems that allow interventions to “fix” problems on the spot without auditing the changes and some that allow software changes after certification of the system.
- ? IRS computer system modernization scrapped after millions were spent but now on its second try with millions more already committed.
- ? John Denver’s airplane where the fuel tank controls were up for off, right for the left tank and down for the right tank.
- ? Systems interacting with one another but using different measurement scales or magnitudes for data, e.g., yards instead of meters; feet instead of miles.

Neumann directed the audience to his web page at SRI (<http://www.csl.sri.com/neumann>) and to the risks site (www.risks.org) for more material. Also, his column in the latest issue of *Communications of the ACM* (September 2004/Vol. 47, No.9) notes major risk categories that he feels face the software industry, in general.

Student Presentations: Empirical Software Engineering (Dewayne Perry, et al)

Dewayne Perry is a UT faculty member at the Center for Advanced Research in Software Engineering (<http://ece.utexas.edu/arise>) and had several students present their work.

Mark Grechanik spoke about a protocol for preventing the replacement of (trusted) system components with others, i.e., the problem of “impersonation” often associated with “spyware.” Some applications run under administrative privileges and can modify anything in the system. Anyone who can make use of access through such applications can add, modify and remove generic components, replacing them with their own versions, which can redirect, for example, data entered through a browser. Therefore, no applications (or users) can be “sure” that they are using trusted components.

Paul Grisham spoke briefly on industry research he has initiated on eXtreme Programming to determine what practices have the most impact and which show the least. [XP, however, is just one of several agile methods. Because of its developer focus, though, it may be one of the easier upon which to base empirical research.]

Matthew Hawthorne spoke on design diversity. The initial problem is that, despite hardware redundancy, running the same software on all the hardware has the same possibility of failure and of limited value in achieving highly reliable systems. Even research on multiple versions of the same software (“n-versioning”) has shown that similar defects may occur when the versions are developed from the same requirements and design (i.e., different versions of the code created by different development groups). Hawthorne is looking at a variety of “diversities” to determine what combinations might produce more reliable systems. Some of the types he is examining are:

- Modal (different ways to accomplish the same functionality);
- Geographical (diverse locations/redundancy);
- Ecological (network diversity);

Temporal (diversity through different timings);

Control (different ways to manage, monitor and control system operations, externally and from within).

Panel: Software Disasters (Dow, Curtis, Perry, Neumann)

This panel brought together people either directly associated with, or who had dealings with people directly associated with, some “well-known” software disasters.

Bill Dow spoke about issues in the development of the Denver Airport. Much of the literature blames the baggage handling system, but it was not just the baggage system. Two delays had occurred before it was even noticed (by United Airlines a full 18 months into the project) that no baggage handling system had even been included in the original requirements. A baggage system was specified with a quote just for United. A year later, it was decided that it would be “unfair” not to provide the same system for everyone else, so it was mandated that it had to be expanded to cover all the airlines with no change in delivery schedule (though more funding was provided). The critical problem at this point was that the software contractor agreed to the extra work without schedule relief. Other difficulties occurred after that. One was that an independent test firm was brought in who was paid by the hour to execute tests of the system and who kept rerunning tests that actually damaged hardware in the system 100 times more costly than the cost of the tests because they would not stop the tests to let the bugs be addressed. Finally, a demo of the system was demanded before it was ready with the press brought in unannounced. Of course, it failed and people had their scapegoat, leading to the reports of the baggage system being “the problem.”

Bill Curtis spoke about the ITT System 12. This huge international telecommunications system involved new software, new staff and a new organization. Many risks related to all this “newness” were not addressed early enough. The system got delivered under budget and ahead of time. However, changes to achieve those goals rendered it impossible to easily (and economically) modify it for all the intended nation-specific target markets. It was sold to Alcatel and the telecom part of ITT went under.

Dewayne Perry spoke on the Ariane 5 (a satellite launch vehicle). Ariane 4 software for trajectory control was reused, but the Ariane 5 was faster and the precision of 4’s software wasn’t up to what 5 needed. A straightforward engineering failure.

Peter Neumann spoke more on a variety of “disasters” due to failure to understand risk issues.

One person in the audience, during the question and answer period, claimed that “*everyone* has known about these kinds of things” (for decades/years). I asked, “Who knows about them?” It was agreed that the majority of people writing software (and managing software projects) do not. A critical element of growing an engineering discipline is learning from mistakes, especially those of others. The software field has not yet developed this critical capability.

Panel: Agile Methods (Krasner, Buhrdorf, Shafer, Duncan)

This panel addressed agile methods compared to more rigorous, formal ones. Herb Krasner and I spoke, ostensibly, in favor of agile methods while Don Shafer and Ross Buhrdorf spoke, ostensibly, about concerns they had in the use of such methods. I say “ostensibly” since there were pro and con comments from all of us, and no one really ended up having anything too harsh to say about agile

methods. Indeed, the main concern expressed had to do with whether those used to very formal contracting and documentation would allow agile methods to be tried, not whether agile methods would be “harmful” in any way.

One audience member said, “We should be doing these things,” referring to many agile methods practices. I replied by asking why we were not even under “formal methods”? For example, Watts Humphrey who championed the development of the SEI’s Software Capability Maturity Model (SW-CMM) started a Personal Software Process (PSP) program because he noted that, in visiting higher maturity organizations, there was still a lack of individual engineer growth in the use of engineering practices. For example, writing unit tests before the code which eXtreme Programming advocates is not found in most formal methodologies/process assumptions. That one would unit test after coding is expected but not that the tests would be written before the code as a guide to what design would successfully pass the tests.

Closing Address: “We have the best people. Why aren’t we winning?” (Bill Curtis)

Using the US men’s Olympic basketball team (since the Olympics were running during the TSSE), Curtis began by noting that “we don’t lose to better talent, but to better teams” (i.e., better organization). Maintaining the sports theme, Curtis described what one track coach said about training a sprinter for the Olympics: “I can’t teach you what God left out.” That is, there is some necessary running ability required compared to the process he could use to teach one to develop that ability, i.e., to use that “natural” ability more effectively. Curtis also referred to Barry Boehm’s new book on Cocomo II and estimation where, after several decades focusing on processes, tools, etc., human performance (both development teams and managers) is still the largest factor impacting project success by 3-4 times.

Heroic effort, unfortunately, is still regarded as a virtue because it becomes demanded by premature commitments, producing work overload, resulting in more mistakes. Some organizations look to outsourcing as a solution; however, that often just displaces an inability to manage development with the hope that the contractor can do it for us. Most organizations have no organizational basis for mature success. If it is, indeed, organization that produces success, Curtis asked “Does your organization add value to your people?” [He cited two books on this topic: *Built to Last* and *Human Capital*.]

But individuals still have to perform and Curtis noted how, in almost all areas, including the arts, people work to master the discipline then develop “originality.” To illustrate this, he showed a progression of paintings and asked what artists painted them. The audience guessed various things and there was general consensus on the different ones right up to the last: Dali’s limp clocks hanging over branches. Curtis then revealed that every painting he had shown was by Dali from a different period in the development of his skill and technique.

Since Curtis has done studies over the years on what represents outstanding design expertise in software development, he discussed some of his findings. For one, a high learning rate and diversity of learning had shown to be more valuable than years of experience (i.e., ten years of programming experience with a single language could be viewed as one year of experience repeated ten times). Other findings were the ability to perform effective information search (from a large store of mental patterns), trying hypotheses (again, often just mentally), and dealing in and between multiple levels of abstraction. This is how an expert works often not knowing or being able to explain the process themselves because trying to explain it (while it is occurring) gets in the way of doing it.

All of these, though, emphasize the need for feedback. This holds true for process effectiveness as well. To do this effectively requires immediate/tight feedback loops. Agile methods, for example, try to tighten the loops and manage uncertainty through onsite customer presence, frequent releases, daily builds and regression testing. However, process discipline is greater when you have less time to wonder what to do next. Therefore, agile methods may require greater personal discipline and skill than formal methodologies which guide and control steps through longer cycles of review and activity, giving people more time to think about what comes next. This is interesting given the criticisms agile methods often receive for being “undisciplined.”

Since Curtis once managed the Process Program at the SEI, which has responsibility for the CMM, he applied the idea of Capability Levels to control in software projects:

Level 1 organizations use “motivation”: “Do it or I’ll find someone who will/can.”

Level 2 ones use planned milestones: what Curtis called “making the world safe for the engineers.”

Level 3 ones use thresholds: milestones with known constraints and knowing “For what we do, what works best?”

Level 4 ones pursue process capability: understand limits with organized tracking/learning.

Level 5 ones address capability gaps: know what they are and have the ability to close them.

Curtis also noted that, as process maturity grows, process-related documentation tends to decrease while the need for tailoring guidelines increases. He also cautioned, though, that capturing experience (through data analysis) was only useful to improve processes when variations are low between examples of practice, i.e., there is a stable process agreed to and followed by everyone.

Successful organizations, Curtis said, display several practices/beliefs:

Learning through peer reviews (not just bug-finding).

A “community of practice” (as existed, Curtis said, at Xerox PARC) where people gather over lunches and managers employ “management by walking around.”

Skill-based mentoring where management feels, “We own the responsibility for how you are prepared.”

Use of a “2-deep chart,” i.e., if the main person on a task is not available, they know who’s next – no lone “heroes” whose absence could sink a project.

In conclusion, Curtis said, there is a progression:

Winners [of a single game] have better people.

Champions [for the season] have better-organized people (especially under stress/change).

Dynasties [who win season after season] have better organizations (i.e., ways to develop people so, despite the players who come and go, the capability is always there).

Final Thoughts

One very nice feature of a conference like the TSSE is the relatively small size of the audience (i.e., <100 people). This encourages a lot of discussion among presenters and audience members. It also results in the roles being swapped from session-to-session, i.e., one session’s presenters will be the next’s audience. And it tends to begin to center discussion and questions around key themes that everyone begins to share as the conference progresses.