Hacker or Hero? — Extreme Programming Today

Steven Fraser (*Impresario)*, Nortel Networks, sdfraser@nortelnetworks.com

Panelists: Kent Beck, First Class Software, 70761.1216@compuserve.com; Ward Cunningham, Cunningham & Cunningham, ward@c2.com; Ron Crocker, Motorola, QA1007@email.mot.com; Martin Fowler, ThoughtWorks, fowler@acm.org; Linda Rising, risingl@acm.org; Laurie Williams, NC State, williams@csu.ncsu.edu

Steven Fraser

Extreme programming is the latest rage, everyone is talking extreme, but who is doing it? XP is in the words of one proponent, is a "lightweight, efficient, low-risk, predictable, scientific, and fun way to develop software". XP bundles much conventional software engineering wisdom into a practice with a high degree of appeal as a cool technology. Questions for inquiring minds include: Will XP deliver? Will XP scale? How will products based on software developed by XP practices age? What are the elements of XP that can be effectively adopted by organizations outside the XP envelop, e.g. large teams, real-time systems, etc. Is XP the next "silver bullet"?

Steven Fraser is a senior manager in Global External Research in the Nortel Networks Disruptive Technology program in Santa Clara, California. Previous to this he was an advisor in technology transfer for object-oriented software development best practices. In 1994 he was a visiting scientist at the SEI in Pittsburgh. Steven completed his doctoral studies at McGill University in Electrical Engineering. He is an avid operatunist, photographer, and videographer.

Kent Beck

I will be taking the anti-XP position, since it clearly cannot work. There are many obstacles in the way of adopting XP, and most of them are completely out of the control of techies. Technically, XP is obvious:

- Evolution of architecture and design
- Constant measurement and refinement
- Frequent automated unit and integration testing

The problem is that once the programmers aren't the bottleneck, aren't, in Ron Jefferies' memorable phrase "the cork in the orifice of progress," all other problems in the business become apparent:

- Marketing really doesn't know what the product should do, and if they make all the priority decisions there is no longer anywhere to hide
- Sales doesn't know how to sell when engineering can tell them exactly where the product is, and can change direction on a dime
- Operations can't keep up with new releases of software daily (for centralized applications) or monthly (for shrink-wrapped applications)
- Executives don't want to make the hard calls about what doesn't go into this release
- Stock analysts can't figure out the additional value of a company where engineering is predictable, fast, and high quality

Kent Beck helped pioneer CRC cards and patterns for software development. He rediscovered test-first programming, and popularized it with his xUnit testing framework, now translated into Java, C++, Visual Basic, and Smalltalk. He is the author of "The Smalltalk Best Practice Patterns", "Kent Beck's Guide to Better Smalltalk", and a contributing author on Martin Fowler's book "Refactoring". He is also the author of the Addison-Wesley books "Extreme Programming: Embrace Change", and "Planning Extreme Programming", and stars with Erich Gamma in the video "Test Infected".

Ron Crocker

I was the system architect for a rather large development effort. The system consisted of several like-sized (10-15 engineers) integrated subsystems. Our project was ideal for an XP approach -the system level approach was to start system integration early, with a little bit of end-to-end functionality, growing the system capabilities until the whole thing was done. Our schedule had three week increments for end-to-end functions, with some shorter increments for pair-wise integration steps. We had a simple system metaphor, easy to remember and meaningful. Not only was I the architect of the system, I was also the development manager for two of the subsystems. The down side was that I only had two of the teams. We did XP (or close to XP - it depends on how literal you want to be), but the other teams were free to do whatever they wanted, as long as they met our incremental integration schedule.

Managing this project became quite “interesting.” Because of our highly integrative approach, we often had stuff that we felt good about. However, because not everyone was playing, we couldn't ship what we had. We also had difficulty telling our management something that they understood. We had a task list for my teams. The task list grew as we went through the planning game. We added tasks as we refined our understanding of what we were doing. We had charts that showed our progress toward achieving our tasks, and we had monotonically increasing number of tasks completed per week, but our “% of tasks completed” number would vacillate based on how many tasks got added each period. We also had data on the integration phases completed. Unfortunately, my management didn't grok this information. They wanted to know if we were done with design, done with coding, done with testing, done with subsystem integration, ready to hand off to system integration, etc. I spent a lot of time trying to convince my management that we were on track, even though it was obvious that we were on track (“Go down in the lab and I'll show it to you”) and the other teams might not be (“We're 93% code complete,” but does that really mean anything?). Ok, so it seems like XP doesn't scale. That doesn't mean XP is bad, it just means that it doesn't scale. There are many projects that don't need to scale to 100s of engineers. XP seems to work well there. We found XP worked nicely for our part of the project, but our approach to building the system this way created quite a bit of different problems. To make XP a scalable approach requires solutions to the coordination issues.

Ron Crocker is a Senior Member of Technical Staff at Motorola. He is responsible for 3rd Generation cellular system architecture, and has been playing with object technology for 17 years. Before joining Motorola, he was a C++ guinea pig at Bell Labs.

Ward Cunningham: *XP is Genius Friendly*

There was a time when a software genius would have to hide from a project to get genius work done. Not so with XP. The genius hangs with the team, taking task cards and pitching in on others. This way a genius gets a good feel for just what sort of invention can be absorbed. To get along, the genius avoids lording superior intellect over others. He/she wouldn't say “I had that idea yesterday”, even if it were true. So what fun is thisThe fun begins when a deep problem surfaces. The whole team feels simplicity slipping away and is distressed about it. Genius gears start working. Should the necessary flash of insight fail to come, the genius just keeps pitching in and making sure that the code stays clean enough to absorb insight that might be just around the corner. The genius knows this wait and enjoys it. Then finally the light comes on. The genius is first to connect the pieces that solve the problem. So what then? Cry eureka? No way. The genius says something like “I'm thinking we need to spend more time looking at the interaction of x and y.” Then the rest of the team has a chance to try their hand at genius. If they are wise enough to look at x and y they will probably see just what the genius has seen and get to bask in the warm light of an original Ah-Ha. The thing that is really great about this interaction is that it is really safe for the genius. If there is an error of logic, it will be gently exposed without casting doubt on the genius. Also, if there is resistance to the insight, the genius gets to explore it before showing all the cards. Finally, the genius is appreciated by people who now know what genius is like. (Remember, they just had a eureka moment themselves.) So what is all this about XP making everyone just a cog? XP is truly genius friendly. And, in case you have any doubt, just think about how the above would work when everyone is a genius. Heck, maybe they are!

Ward Cunningham is a founder of Cunningham & Cunningham, Inc. He has also served as Director of R&D at Wyatt Software and as Principle Engineer in the Tektronix Computer Research Laboratory. Ward is well known for his contributions to the developing practice of object-oriented programming, the variation called Extreme Programming, and the communities hosted by his WikiWikiWeb. He is active with the Hillside Group and has served as program chair of the Pattern Languages of Programs conference which it sponsors. Ward created the CRC design method which helps teams find core objects for their programs. Ward has written for PLoP, JOOP and OOPSLA on Patterns, Objects, and CRC.

Martin Fowler

XP is an interesting methodology. It calls itself lightweight - but it's high discipline means that it's actually quite difficult to adopt. People talk about doing it, while missing out some of the practices. Others don't even claim to do it, but may use as many of the practices as those who do - sometimes they may say they are doing pseudo-XP. There is no conformance test (or merit badge) for XP, so how can anyone tell if they are doing it or not? For some this is a problem, but frankly I don't mind that much. The key to XP, at least for me, is that it has acted as a cattle prod to the rather staid word of methodology. It's forced people to reassess what it means for software to become a more disciplined profession, and it's reintroduced a number of important practices to software development that have long been neglected. Even without the rest its emphasis on testing, balanced approach to planning, and enabling of evolutionary design are big steps forward. In the end you can't choose a methodology and drop it into a project. Instead you have to start with your company culture, your type of project, and above all your people, and blend a methodology that will work for those circumstances. For many situations (up to a dozen co-located developers with uncertain or volatile requirements) XP

makes a good starting point. But it can only be a starting point.

Martin Fowler is Chief Scientist at ThoughtWorks, an internet system integrator, where he is responsible to identifying and spreading best practices for ThoughtWorks and their clients. He's been active in object circles for over a decade, developing business software in C++, Smalltalk, and Java. He's the author of Analysis Patterns, UML Distilled, Refactoring, and soon Planning Extreme Programming. He first learned about Extreme Programming from Kent on the Chrysler C3 project, and has tried to apply lessons from that experience ever since.

Linda Rising

I know XP is hot! I've recently given several talks on eXtreme Programming - at the joint IEEE-ACM-SPIN meeting in Phoenix, the attendance was triple that of our usual gathering. Afterwards, the Tucson and Flagstaff organizations wanted the same presentation. There's definitely interest out there! Since I've been spending some time in the trenches, I tried to introduce XP to my team. They liked the sound of it but implementing it was another story. Kent says himself in his book that XP sounds a lot easier than it is. I was also a member of the program committee for the first ever XP conference and I know there were lots of paper submissions. In the ones I read, however, I didn't see anyone who was actually practicing full-blown XP. They were touting the benefits of one tenet or another but my question still remains: is anyone (except Kent et. al.) really doing it?

Linda Rising has a Ph.D. from Arizona State University in the area of object-based design metrics. Her background includes university teaching experience as well as work in industry in the areas of telecommunications, avionics, and strategic weapons systems. She has been working with object technologies since 1983. Linda Rising is the editor of: The Patterns Handbook, A Pattern Almanac 2000, and Design Patterns in Communications Software..

Laurie Williams

XP popularized the practice of pair-programming – *two* programmers working side-by-side at *one* computer, collaborating on the same design, algorithm, code or test. XP has touted near defect-free code production without increased costs, which has peaked the interest of many – including myself. Still, many software development managers question the affordability of the practice. Convention speaks against having two people work together to develop code -- having “two do the work of one”, as some people see it. In 1999, a structured experiment was run at the University of Utah to study pair-programming. The results of the experiment showed that, indeed, pair-programmers produce code with statistically significant higher quality and that the cost increase for “two doing the work of one” is not statistically significant. Additionally, employing the practice of pair-programming can help an organization achieve a higher level of maturity. Pair-programmers tend to apply a positive form of “pair-pressure” on each other. This pressure causes them follow more faithfully the prescribed methodology, whether it is XP, PSP, RUP or any other. Based on surveys and direct experiences, programmers are far less likely to “blow off” writing test cases or doing design if they are working with a partner. (They are also less likely to read their email or surf the web!) Achieving widespread acceptance and adherence to a methodology is often a major inhibitor to organizations trying to realize the increased effectiveness and predictability that is a major benefit of maturity. Lastly, almost unanimously, pair-programmers have said that they enjoy working with a partner and that they feel more confident when working with a partner. Therefore, pair-programming is not likely one of those process improvements that gets “forgotten” in crunch time. Quite commonly, programmers, long conditioned to working alone, initially hesitate in their transition to pair-programming and then buy-in after trying it. Sporadically, a pair-programming arrangement does not work out due to the personalities involved. Most often, the pair and the organization achieve the quality improvement and maturity that have been described.

Laurie Williams is a faculty member at North Carolina State. She received a BS in Industrial Engineering from Lehigh University, an MBA from Duke University, and a PhD in Computer Science from the University of Utah. Laurie worked at IBM in Research Triangle Park, NC for nine years in engineering and software development technical and managerial positions. In her last position, she managed a software-testing department. Her research interests are in software engineering, software process, collaborative programming and eCommerce.