TWENTY-FOURTH ANNUAL PACIFIC NORTHWEST SOFTWARE QUALITY CONFERENCE

October 10 - 11, 2006

Oregon Convention Center
Portland, Oregon

2006 CONFERENCE PLANNING COMMITTEE

Randal Albert

Rick Anderson
Tektronix, Inc.

John Balza
Hewlett Packard

Kit Bradley
Wildblue Consulting

David Chandler
Nationwide Insurance

Albert Dijkstra

James Fudge
McAfee, Inc.

Dennis Ganoe
Optilink Healthcare

Brian Hansen

Jason Kelly
Microsoft

Randy King

Jonathan Morris
Kronos/Unicru, Inc.

Sudarshan Murthy
Portland State University

Ganesh Prabhala
Intel Corporation

Ian Savage
McAfee, Inc.

Eric Schnellman
JanEric Systems

Wolfgang Strigel
QA Labs Inc.

Ruku Tekchandani
Intel Corporation

Richard Vireday
Intel Corporation

Scott Whitmire
Progressive Solutions (USA), Inc.

Yabo Wang
Hewlett Packard

Evolutionary Methods (Evo) at Tektronix: A Case Study

Frank Goovaerts, Tektronix Inc., (503) 627-2224, frank.h.goovaerts@exgate.tek.com
Doug Shiffler, Tektronix Inc., (503) 627-7588, douglas.e.shiffler@exgate.tek.com

Biographies

Frank Goovaerts has engineering degrees from the University of Leuven (Belgium) and the University of Cincinnati.
He has worked in the software industry since he graduated in 1981. For the last 16 years he has been with Tektronix in a number of roles varying from developer, to Project Lead, Functional Manager and currently Director of Engineering for the Performance Oscilloscope Product Line. Tektronix is a world leader in test, measurement, and monitoring equipment.

Frank is an advocate for process improvement and continues to drive his organization to create better software, on time.

Doug Shiffler has an engineering degree and a business degree from Brigham Young University. He has worked in the computer and test and measurement industries for the past 23 years in a variety of roles in process, design and quality engineering, program management and is currently a Senior Engineering Manager in the Performance Oscilloscope Product Line.

Doug is a quality advocate and is a believer in the benefits of continual process improvement.

Abstract

This paper documents how we implemented the Evolutionary Method (Evo) at Tektronix, Inc. We start with a brief refresher on this method that dates back to the 1980's (Deming cycle). The method was then evolved by Tom Gilb in 'Principles of Software Management' in the late 80's. We discuss the development methodology used at Tektronix and how the Evo method fits into that methodology. We then detail a case study where we implemented Evo mid project (in a large team) and tracked intermediate results up to the end. We conclude with lessons learned and where we plan to go next.

1 Intro

One cannot argue the fact that the production of high quality products results in a competitive advantage, especially for a company like Tektronix, a world leader in test, measurement and monitoring. In fact, our quality policy states that "Tektronix is committed to providing products and services that consistently meet or exceed our customer's expectations, while continuously improving the effectiveness and efficiency of all aspects of the operation". However, quality comes at a price. There are numerous references (Wiegers, McConnell and others) to the fact that defects take up to 200 times longer to fix later in the lifecycle. Still, we end up fixing those pesky defects at the end of the program, despite the expense (causing both time and budget overruns).

We deployed Evo to help us plan and track milestones and to reduce product development costs by focusing on fixing defects earlier in the product lifecycle.

The case study involves a major platform development effort including software and hardware. One of the strengths of the method is that it can be rolled out across an organization or can target a subset of that organization; it works on small programs as well as larger ones. We initially tried Evo with a small team of software engineers; we then rolled it out to the rest of the software team, and finally deployed it to other functional organizations. Although we cannot claim the success of this program to using the method (after all we didn't run another program where we did not use the method in parallel), we believe that Evo helped us get the product out in time with excellent quality. Most of the team members agree that Evo helps them with day to day prioritization and team communication.

We will explain how we adapted the Evo method to our projects and product development lifecycle. One of the essential strengths of the method is that it forces the project to evaluate frequently and make changes where needed.

2 Evo Explained

Evo is an abbreviation for the development of an **evolutionary** delivery process as a means of program management. Rather than use the traditional waterfall approach to managing product development which emphasizes one large delivery at the end of the complete development cycle, Evo focuses on multiple mini-cycles. Each cycle is concentrated on a subset of the end requirements. It focuses on a prioritized list of what the stakeholder/customer needs are at a given point in time to the level of detail that is required at the time.

Tom Gilb who is an authority in Evo techniques has defined 10 Evolutionary Project Management (Evo) principles. This information and much more can be found on Tom's web site at http://www.gilb.com.

Evo Principles (paraphrased from Gilb's web site):

1. Deliver real results, of value to real stakeholders, early and frequently.
2. The next Evo delivery step must be the one that delivers the most stakeholder value possible at that time.
3. Evo steps deliver the specified requirements, evolutionarily. Concentrate on delivering real value to the stakeholder (as defined by the stakeholder) and then measure your results by asking the questions:
    - Exactly what value was delivered?
    - Exactly what did it cost compared to estimates?
    - Was the stakeholder really satisfied?
    - Did new requirements get discovered?
    - Did the technology work as expected?

    Evo is about feedback and learning. It is about applying multiple 'Plan-Do-Check-Act' (Deming) cycles concurrently.

    One important consequence of this is that the formal requirements are very important. If they are not unambiguously clear, if benefits and costs are not quantified – then we cannot use Evo in the rational engineering mode that is the expected mode of use.
4. We cannot know all the right requirements in advance, but we can discover them more quickly by attempts to deliver real value to real stakeholders.
5. Evo is holistic systems engineering; all necessary aspects of the system must be complete and correct and delivered to a real stakeholder environment. It is not only about programming, it is about customer satisfaction.
6. Evo projects typically require an open architecture, because we are going to change project ideas as often as we need to, in order to really deliver value to our stakeholders.
   Open architecture means 'easy to change'. This can include many 'technological enablers', for many types of change. If you want to optimize your product for long term performance (and consequent survival) under conditions of change, you have to be conscious about your product architecture objectives and design specifications. When the stakeholders define a requirement we have to be able to deliver those requirements immediately without compromising system performance. That is what we need the open architecture for. All systems need intelligent open architecture in the long run. Evo projects need it in the short term, during the project.

7. The Evo project team will focus their energy, as a team, towards success in the current Evo step. They will succeed or fail in the current step, together. They will not waste energy on downstream steps until they have mastered current steps successfully.
8. Evo is about learning from hard experience, as fast as we can – what really works, and what really delivers value. Evo is a discipline to make us confront our problems early – but which allows us to progress quickly when we really probably have it right.
9. Evo leads to early, and on-time, product delivery - both because of selected early priority delivery, and because we learn to get things right early.
10. Evo should allow us to prove out new work processes, and get rid of bad ones early.

3 Development Methodology

3.1 Find and Fix Defects Sooner

A key part of our development Methodology is to "Find and Fix Defects Sooner" in the development lifecycle. Ultimately this can lead to defect prevention and eventually defect free code. Our definition of defect free is no defects that will prevent the customer from using the product to its specifications.

The entire industry is in agreement with the fact that having to fix defects later in the life cycle costs a lot more. Multiple resources can be referenced, here are just a few:

- 10-100 times as much to correct once fielded (Steve McConnell, 2001)
- 50-200 times as much to correct once fielded. (Barry Boehm, 1988)
- 15 times as much to correct once fielded (IBM System Sciences Institute)
- 10 times as much to correct during testing (Hughes Aircraft)

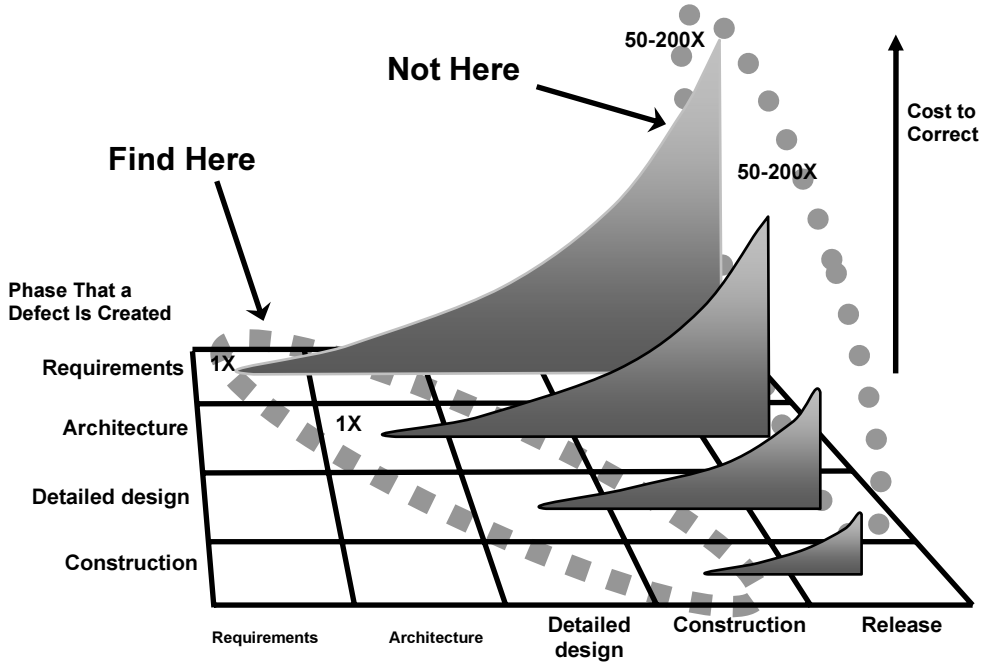
On most software development projects, reworking requirements defects alone costs an inordinate amount of resources:

- 40-50% of the effort (Capers Jones)
- 80% of the effort (Karl Wiegers, 2001)

Finally, up to half of all project resources are used for general 'rework', this includes changing requirements, fixing introduced defects, and re-architecting the code.

The following graph is taken from one of Steve McConnell's presentations. It illustrates how the cost to fix defects can skyrocket during the development lifecycle.

Living with Defect-Cost Increase



Source: Steve McConnell

3.2 Mini-Milestone Development

In support of "Finding and Fixing Defect Sooner", we also institutionalized mini-milestones as part of our Development Methodology.

Frequent deliverables of 'finished' products seems to be the norm in today's Agile world. That doesn't fit for a Test and Measurement company that delivers complex, high quality equipment and software to its customers. To allow us to deliver these quality products (that meet published specs) to our external customers, we follow a stringent waterfall based New Product Development methodology. In other words, we decide what to do, how to do it, do it, test it and ship it. And the 'do it' cycle typically takes the longest.

We tailor the mini-milestones to internal customers. This allows us to release 'finished' products to other organizations which include hardware and software engineering, documentation, qualification and others. The goal is to build excitement for these intermediate releases, both within the development team and the internal customer.

While focusing on mini-milestones, it is important to not lose sight of the final deliverable (i.e. the finished product to our external customers). This is tracked by the program timeline which contains (at a high level) a list of all features and estimated effort (preferably prioritized) that need to be completed to meet the product specifications. Each of these features needs to be planned out with enough details so we can make an accurate timeline, after all we need to feel comfortable that we will deliver a product to our external customers within the defined timeframe.

The deliverables for each mini-milestone need to be clearly defined to allow the team to self measure. The list of deliverables needs to be realistic. Adding deliverables that have no chance of completion adds no value. Ideally, each team member should have one or more deliverable

listed. For each (small) incremental delivery the team can judge how well they did and adjust accordingly. That may include cutting features (better to do that early on, before development starts), adding resources, changing the timeline (if the market allows), or any other proactive (rather than reactive) measure we choose to take.

4 Case Study

4.1 Expectations from this project

This was a multi-year project to develop a new high-end oscilloscope platform. Historically our experience has shown that the software portion of a new product development program is often delivered late. After reviewing the Evo literature and examining the results from others who have used Evo, the authors believed that the implementation of Evo techniques on this program could decrease development time and improve software quality. Evo literature suggested that the use of Evo techniques would result in about 30% schedule improvement as well as higher product quality.

4.2 Background

This program was partially through the engineering development lifecycle when we decided to implement the Evo techniques. The software was developed by a large distributed team of software engineers. The software development was dependant upon the timely delivery of working hardware. Delays in the hardware delivery could affect software development schedule day for day.

At the time of Evo implementation the hardware and software schedule was developed using the traditional waterfall approach in Microsoft Project. A top level schedule was maintained by the program manager and the software schedule (that dovetailed into the program schedule) was maintained by the software project lead. Weekly schedule update sessions were held by the software project team leader and members of the software development team followed by project reconciliation sessions with the program manager.

The challenge for us was to implement a new product development management methodology (mid-project) that would positively improve the schedule and performance of the software team. At that point in time, only the software organization had agreed to use the Evo techniques, the other functional areas chose not to implement Evo.

Evo implementation challenges included:

- Implementation of Evo across a distributed team.
- The software organization was the only functional organization to implement Evo techniques for this program.
- We implemented Evo mid-program rather than at the beginning of the product development cycle.
- Neither management nor members of the development team had prior experience using Evo techniques.
- Almost everyone was skeptical about the implementation of a new process mid-program.
- Many members of the software development team were experienced engineers and therefore used to the traditional ways of planning and executing their software development tasks.

4.3 Short Term Goal

The short term goal was to demonstrate verifiable schedule and quality improvement over historical averages obtained on past product development programs.
The actions taken to achieve this short term goal were:

- Investigate Evo techniques and assess the potential benefits to our software development processes.
- Utilize the knowledge gained from this research to explain and request the support of the General Manager, Program Manager, Software Project lead(s) and the individual engineers.
- Train the software engineering team on Evo techniques with the assistance of Niels Malotaux, an Evo coach based in the Netherlands.
- Tailor the method in such a way that we could adapt it mid-way through the development cycle.
- Determine how to link the Evo weekly scheduling into the master project schedule.
- Determine metrics to use to track progress and adherence to the Evo techniques.
- Minimize the impact on the developers as to not impact current activities.

4.4 Long Term Goal

The long term goal was to realize a 30% improvement in schedule pull-in and quality improvement over the conventional waterfall method.
The actions taken to achieve the long term goal were:

- Demonstrate sufficient benefit to expand the use of the methodology to other functional groups.
- Roll Evo techniques out to other functional groups to realize even greater benefits across the organization.
- Continue to utilize and refine the Evo techniques.

4.5 Evo Implementation Time Line

We implemented Evo over a period of 8 weeks. A detailed chronological report on how this method was introduced in our organization follows.

4.5.1 Week 1

- The general method was explained to the entire organization, not detailing any implementation plans.
- Software engineering management then presented the proposal to stakeholders. Evo was approved for use on a trial basis for the software development portion of the program.
- We held an initial training session with the Beaverton software team.
- Due to the critical nature of this program, the team was given multiple opportunities to decline using this method.
- We started a four week cycle on a trial basis with a subset of the team (8 developers). At that point the team had made a commitment to use the method during this period (reason: it's easy to get overwhelmed by the perceived initial overhead, after three to four weeks people had adapted to using the method and were seeing the benefits).

4.5.2 Week 2 through 5

- The team used Evo on a weekly basis. Niels Malotaux guided us through the initial one-on-one meetings as well as team meetings
- We evaluated the experience of the Beaverton SW development group and decided that we would continue to use the technique
- During these four weeks, we implemented 62 out of 70 features on time!
- The process was rolled out to the Bangalore-India development group for another four week trial run to determine whether Evo would work as well with a remote team. If that effort was successful, we would roll out the method to the entire development team.

4.5.3 Week 6 through 8

- The Beaverton team continued to use the method
- The Bangalore team started to use the method.
- We devised a way to synchronize Bangalore/Beaverton Evo activities by setting up a weekly conversation between the team leader in Bangalore and the liaison in Beaverton.
- Results where favorable

4.5.4 Week 9 through Project End

- We included the entire software team (unconditionally)
- Continued weekly Evo reporting and planning sessions with all members of the development team.
- The team was rather large, so Evo team meetings and reporting were divided into sub-teams. This caused us to need a roll-up meeting.

4.6 Evo Implementation Guidelines

The following were the guidelines that were established for executing the Evo process within the Software team. These guidelines were meant to be a quick reference for members of the team for the most relevant aspects of the process. The team also utilized and referenced the handbook, "*How Quality is Assured by Evolutionary Methods*" (Niels Malotaux),

1. Weekly Evo One-on-one Sessions

These sessions are held between sub-project leaders and the engineers assigned to the sub-team. These sessions are also held between sub-project leaders and the software project lead. Functional managers also attend these one-on-one sessions on a rotating basis to lend support, clear obstacles and to ensure the process was being followed. Guidelines for these sessions included:

- Duration of the one-on-one sessions should be brief (less than 15 minutes).
- Developers should come to the session prepared with a complete list of tasks for the upcoming cycle.
- Developers come prepared with 26 to 30 hours of proposed work (the remaining hours are ‘unplanned’).
- If a developer cannot identify a full 26 hours of tasks, they should discuss it with their subproject lead prior to the Evo one-on-one session.
- Any issues or action items identified during the session should be noted, addressed, and tracked outside the session.
- Long discussions on design ideas, bug evaluations, etc. should be avoided.

2. Evo Task Planning

- Two thirds of total working hours should be planned (26 hours for a 40 hour cycle).
- Maximum task duration should be six hours.
- Task estimates should include only actual hours spent working on the task (effort hours), not overhead or interruptions.
- Planned absences should be deducted from the total working hours, (e.g. if a developer is working 32 hours in a cycle, plan two thirds of the 32 hours or ~20 hours.
- When an unfamiliar task is encountered, an analysis task should be in addition to the other tasks. After the analysis task has been completed the developer and subproject lead can finish the cycle plan or plan it out in the next cycle.
- Dependencies which affect the ability of an engineer to perform the next cycle's tasks should be noted and confirmed with the individual responsible for delivering the dependency prior to the Evo one-on-one sessions.
- Missed one-on-one sessions should be made up as soon as possible with the subproject lead.
- Each developer should keep a hardcopy of the current cycle's tasks.
- No changes should be made to the Evo database outside of the one-on-one sessions (with the exception of marking a task done, adding details to current tasks and entering ideas for the next cycles(s)). The tool does not allow us to track this automatically, so we rely on people following the process.
- One should schedule the most important task first (as determined by the Project Lead, who in turn gets this input from Core Teams and change control boards).

3. Evo Task Completion

- All tasks for each cycle should be completed as planned.
- If for any reason a planned task cannot be completed during the cycle, the developer should notify the subproject lead as soon as possible. The developer and lead should then re-plan as necessary. Changes to the plan should be recorded on the developer's task sheet.

4. Evo Tracking

- The subproject and project lead must be able to link and track the Evo tasks (hourly/daily resolution) to the master program schedule (weekly resolution) and the milestone checklist.

4.7 Evo Results

4.7.1 Evo near Term Results

Schedule accuracy for this platform development was 50% better than the program average (as measured by program schedule overrun) over the last 5 years, and this product was the fastest time-to-market with the highest quality at introduction of any platform in our group in more than 10 years. The team also won a prestigious Team Award as part of the company's Technical Excellence recognition program.

4.7.2 Evo Sustainability

Based upon the results realized during this program there is across the board support for the use of Evo techniques on the follow-on program both in software and hardware. We believe that having both of these groups utilizing Evo techniques will provide even greater schedule predictability because of the dependencies between the software and hardware schedules. Higher reliability in the hardware schedule will provide higher reliability of the software schedule.

5 Lessons Learned and Next Steps

Following are some of the lessons we have learned after using this method for almost two years now. We intend to use these to fine-tune the process and continue to improve the method.

- Identify early in the project who are the Stakeholders in the Evo process and document expectations.
- Schedule and hold Milestone review meetings in order to gauge the plan to the actual execution of the program.
- The initial implementation of Evo began with the Software Engineering team and we believe that improved results will occur with the inclusion of the Hardware Engineering team in the Evo process during the next development program. This will more clearly highlight as well as track the dependencies between the two functional areas.
- At the start of the Evo process build excitement for stakeholders so that they anticipate and validate the Evo deliverables.
- The first implementation of Evo was implemented mid-way though the product development program, improved results will occur as Evo is implemented at the start of all future development programs within our product line.

5.1 Evo results we have witnessed and the drivers behind them

Even though this paper only talks about the implementation of Evo in the software organization of one product line, we have seen many benefits. These include:

- Better quality end-products through improved cross functional integration.
- Constant optimization of resources through continual feedback in the Evo process.
- Reduced rework through emphasis on doing work right the first time (i.e. no defects).
- Completion of the highest leverage tasks first through prioritization (postpone lower priority or less impact tasks). These may include user interface paradigms that we may want to validate, architectural risk reduction activities, manufacturability issues and others.
- Creates a sense of urgency in the team through regular review of performance to plan.
- Feeling of accomplishment as team members are encouraged to have goals and the feeling of accomplishment (or failure at times) as they measure against their goals.
- Timely corrective action through early and continuous feedback on individual contributors to Project Leads and functional managers. (Roadblocks, scheduling inaccuracies become obvious right away).
- Better efficiency through the use of Time-boxed development which keeps people from getting ‘stuck’ on a problem.
- Improved project planning (Makes it obvious when we have poorly planned providing the motivation to take steps to improve).

The drivers behind these results can be summarized as follows:

- Everyone works on the most important things all the time, priorities are evaluated weekly.
- Promotes the team effort and increases leverage (rather than a group of individuals doing great work).
- Project leads are more closely involved with all aspects of the program.
- Project leads are aware of any issues that arise during any given day.
- Individual productivity is improved because people stay more focused.
- Improved team dynamics during the weekly Evo Team meetings.
- Evo is all about questioning what NOT to do, so that we have more time to do the things we really have to do well.

5.2 Summary Observations

Overall takeaways from using Evo in our organization:

- Evo is helping to bring forth some best practices that will help our engineering development effort.
- Our goal is to take the best parts of Evo and utilize them. This will bring additional best practices to the table.
- During this first implementation of Evo we went through the mechanics of the delivery cycles, not always building the excitement with the team as well as the (internal) customers. In the next use of Evo we need to identify the internal customers up front and ensure we understand and document those items that build excitement.
- We implemented this method mid project (execution phase), we need to adapt this method to the rest of our product development lifecycle, lifecycle (i.e. the definition and design phases) and include the Hardware engineering team in this process.

6 References

Gilb, Tom: Principles of Software Engineering Management, 1988, Addison Wesley, ISBN 0-201-19246-2
Gilb, Tom: Competitive Engineering, 2005, Elsevier, ISBN 0750665076
W.E. Deming: Out of the Crisis. MIT, 1986, ISBN 0911379010
McConnell, Steve: Rapid Development, 1996, Microsoft Press, ISBN 1-55615-900-5
McConnell, Steve: Professional Software Development, 2004, Addison Wesley, ISBN 0-321-19367-9
Malotaux, Niels: "*How Quality is Assured by Evolutionary Methods*",
http://www.malotaux.nl/nrm/pdf/Booklet2.pdf
Malotaux, Niels: "*Evolutionary Project Management Methods",*
http://www.malotaux.nl/nrm/pdf/MxEvo.pdf

http://www.gilb.com
http://www.malotaux.nl/nrm/Evo