JOIN OVER 100,000 PROS! Join Log In | Member Benefits

STICKYMINDS

Agile Testing DevOps Security AI

The Unchangeable Rules of Software Change

ARTICLE

By Steve Berczuk , Brad Appleton Robert Cowham | February 14, 2006

SUMMARY

The authors examine the fundamental truths about project and requirements changes: What can we control? What is beyond our control? What are some of the common perils and pitfalls of change-control and iterative development? We discuss how to avoid many of these common pitfalls without creating new ones along the way, and provide a wealth of resources for first-timers to iterative development.

I this article, we examine the fundamental truths about project and requirements changes: What canf we control? What is beyond our control? What are some of the common perils and pitfalls of change-control and iterative development? We discuss how to avoid many of these common pitfalls without creating new ones along the way, and provide a wealth of resources for first-timers to iterative development.

Pitfalls of Controlling Change

I often come across teams in the early stages of learning to deal with changing requirements. They typically run into two pitfalls in the following order:

Pitfall #1: No scope change-management

The very first pitfall they fall into is not having any kind of change-management for the project/requirements. They allow any and all changes in a very reactive fashion without thinking about trying to renegotiate the scope, schedule, or resources for the project/release.

Pitfall #2: Preventing scope changes

The very next time around, they run into the second pitfall: they overcompensate for being burned/bitten by the first pitfall by trying to prevent any and all changes to the

We use cookies to ensure you get the best experience on our website.

Accept

Read More

They keep insisting that the fundamental problem is they don't have completely stable detailed requirements. If the requirements were detailed enough up-front, they think their estimates would be more accurate; and if only the requirements would stop changing before a single line of code is written, then the schedule wouldn't keep fluctuating so much. It's those darn users/customers/analysts that don't know what exactly they want at the outset. It's all their fault!

This leads to another pitfall:

Pitfall #3: Requirements Analysis Paralysis

They spend inordinately more and more effort analyzing and engineering requirements and specifying them in far too much detail before attempting to validate their results with tangible feedback.

The irony is that the more they try to get stable detailed requirements up-front, the more the schedule becomes protracted: first to get all the gory-details analyzed and specified up-front; and second because so many of the gory-details were either incorrect or still not detailed enough. It becomes this vicious cycle of trying to prevent change with more up-front detail, and yet things keep getting worse instead of better. If the cycle goes unrecognized and unchecked, it can turn into a "holy grail" crusade in search of the perfect and unchanging requirements.

Biology, Stability, Humanity, Uncertainty, and Inevitability

When I encounter a group in the midst of these pitfalls, the first thing I commonly do here is explain the following:

There is a very fancy technical term that biologists use to describe completely stable systems. This highly sophisticated technical term is the word "DEAD!"

I then try to explain that we meager humans (including ourselves and our customers) are imperfect, and we have imperfect and incomplete knowledge: We don't know things, and we don't know that we don't know things, and we don't know how to find out many of those things earlier.

Then I wax philosophically and authoritatively explain the unalterable truth about the fundamental nature of software development and changing requirements. I call it my "tried and true, battle-proven and industry-acknowledged, unchangeable rules of software change ...

The Unchangeable Rules of Software Change

Rule #0: Change is Inevitable!

The Requirements/Plans ARE going to change!

Rule #1: Resistance is Futile!

There isn't a darn thing you can do to prevent Rule #0.

We use cookies to ensure you get the best experience on our website.

The more you try to deny and defy rule #1 by attempting to prevent rule #0, the worse things will get.

Rule #3: Change is like Quicksilver -- Tightening your grip makes it slip out of control

The more you try to deny and defy rule #2 by attempting to precisely predict, or rigidly control change, the more erratic and onerous the result will be.

Rule #4: Embrace change to control change

The more flexible and adaptive you are at accommodating change, the more control you will have over your outcomes.

These rules are very hard to accept! Often, they are accepted only one at a time, after much frustration and over a period of hard learning and reflection. The stages one passes through while learning these rules bears more than a passing resemblance to **"the five stages of grief."**

Traditional Change-Control

When faced with a requested or impending change that impacts the scope of the project, generally accepted best-practice is to utilize a Change-Control Board (CCB) consisting of your project's key stakeholders:

- Assess the impacts of the change make those known to your stakeholders
- Present the viable options the risks associated with each one (possibly accompanied by a recommendation)

- Let them decide which option to "exercise" based on informed consent, making sure they are fully cognizant of the benefits, consequences, and risks of the chosen option.

Still, the idea of giving up control to another group every time a change arises isn't always palatable. And of course there is also the fear that every a scope change that becomes visible will make them look bad, as if they don't really know what they are doing.

Recently I was talking to a group that was struggling with rule #2. They thought if they could only do even more detailed specification up-front (they already do a reasonable amount of up-front detail), that it would somehow eliminate problems with estimation accuracy, which in turn would alleviate problems with "conformance to plan" and prevent the details from being discovered later (because they would instead get them all "right" up-front).

Despite having plenty of historical evidence/data in this particular product to support the "inescapable truth" laid out by these rules, there still seemed to be that desire to cling to the illusion of control that we can somehow prevent such changes if only we spend more time & effort getting a more complete, perfect & detailed specification up-front near the very beginning.

We use cookies to ensure you get the best experience on our website.

looked for some materials from other folks more sagacious and experienced than myself. The first that came to mind was **Lehman and Belady's Laws of Software Evolution** (ca.1971) and the subsequent work on **the FEAST Projects**. Those are primarily about software maintenance however, and refer to the inevitability of needing to maintain and evolve an already released piece of software. They don't speak quite as strongly to the inevitability of changes via feedback, discovery and learning during the course of any non-trivial software development effort.

So I searched a bit more and came across a handful of particularly interesting (to me) references that are quite recent:

- Phil Armour's Jan 2006 CACM article Counting Boulders and Measuring Mountains: Accuracy and the Measurement of Peaks and Projects (Phil's book **The Laws of Software Process** is one of my favorites)
- Chapter 2 of Karl Wieger's newest book **More About Software Requirements** (Microsoft Press, 2006; ISBN 0-7356-2267-1) entitled **"Cosmic Truths About Software Requirements"**

- A recent gem of a posting from **Per Kroll**, Director at Rational Software and co-author with Phillipe Kruchten of "**The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP"**, posted on the IBM/Rational "RUP" discussion forum on Jan 6, 2006:

The fundamentally difficult hurdle managers need to come across is that when you do not know enough to produce an accurate estimate, it will not help to do a more fine-granular plan. Rather, you need to do a list of what are the unknowns and risks preventing you from doing a more detailed estimate, and then rapidly do the analysis, design, implementation, testing, or other risk mitigation turning unknowns into knowns. This sounds as something that should not be very difficult to understand, but it really is a big mountain to climb for many people...

Iterative Development to the Rescue?

One of the things the particular team finally figured out it needed was to try and use iterative development as a means of managing change and being more responsive to customer feedback while still controlling scope:

- One of the ways in which iterative development is most effective at controlling change is that it shortens the time-window of opportunity during which changes can occur!
- The other is that it shortens the cycle-time of the feedback-loop for obtaining validation of our end-result. The sooner we can see tangible executable and obtain stakeholder feedback, the more we reduce the risk of unmet expectations and the associated cost of rework!

We use cookies to ensure you get the best experience on our website.

often do next. Well, as you might have guessed, their first attempt at iterative development doesn't quite work as they expected. They often run into three more pitfalls:

Pitfall #4: Insufficient Requirements Analysis

Thinking iterative development the "silver bullet" to their requirements problems, they abandon all or most of the requirements analysis they had been doing, and don't understand the requirements as well as they should before starting to code.

Pitfall #5: No iteration planning

They jump into iterative development in an overly ad-hoc fashion, without a suitable iteration plan for each iteration. They naively assumed "going iterative" would eliminate the need to define specific goals and milestones for each iteration at the inch/pebble level.

Pitfall #6: Too much iteration planning

Once again, on the heels of their previous failure, they go too far to the other extreme and try to apply all the formality and rigor of planning a large project to the planning of each iteration. They burden themselves with an amount of project management and tracking effort that was intended to take place over many months and try to squeeze all of it into a period of a few weeks.

Walking the Tightrope and Avoiding the Pitfalls

I've seen teams that ran into all of these pitfalls as well. The remedy for these three latest pitfalls is not necessarily to fall back into trying to overanalyze and overspecify detailed plans and requirements on a per-iteration basis. Some initial amount of planning and requirements is necessary to know where you are trying to go and what to estimate. The remedy is to stop bouncing back and forth between extremes, and instead of doing all or nothing "up front," do a reasonable initial amount and then incrementally and iteratively flesh out each individual piece, instead of trying to do scores or hundreds of partially completed pieces all at the same time.

Good change and configuration management is supposed to be about facilitating change, not about preventing it. That's what the **Agile Manifesto** really means when it talks about "Responding to Change over Following a Plan." The point isn't to discard or dismiss plans or planning, it is to deal with and adjust to reality.

As Winston Churchill once said "Plans are of little importance, but planning is essential" -- Dwight Eisenhower said it much the same way: Plans are nothing; planning is everything!" But conforming to reality must takes precedence over conformance to "the plan" (trust the terrain, not the map!). More recently, **James Highsmith** wrote: "Agile projects are not controlled by conformance to plan but by conformance to business value.... If we accept the notion of constant change and turbulence, then plans are still useful as guides, but not as control mechanisms -- because they tend to punish correct actions."

We use cookies to ensure you get the best experience on our website.

them!"

How does all this relate to traditional CCBs and how does it change "change control" for agile and iterative development? It means that:

- Plans and requirements are still necessary to identify, but should be elaborated iteratively in progressively more detail, usually a feature at a time.
- Regular (e.g., at least weekly) milestones are still important to set and track -- don't forget to do that just because you're working in short, fast increments
- Incremental integration needs to be even more frequent (and automated) when iterations are short
- Manage stakeholder expectations with close communication and simple boundaries (short, frequent iterations)
- If iterations are very short, there is little opportunity to work on anything but the agreed upon features for the current iteration. In this case, CCBs are really just iteration planning meetings with your stakeholders at the beginning of each iteration.
- CCB's and related decision-makers must be more dynamic, comprised of stakeholders that are readily available and accessible to give rapid response to issues and questions soon after they arise.
- At the start of each iteration, expectations and priorities are (re)set and (re)calibrated

Iterative Development Resources:
Of course, saying "do iterative development" is one thing. Figuring out how to actually do it for a group in an organization that isn't accustomed to it is another thing entirely. So here is a list of resources on the subject of adopting, planning/managing, and doing iterative software development -- particularly for those coming from a background of phased-sequential (waterfall or "V") model of planning:

- **Wikipedia page on Iterative Development**
- Also from Craig Larman, **Iterative and Evolutionary**
- **Overcoming cultural challenges in adopting iterative development**, by Clay Nelson, The Rational Edge, October 2004
- Philippe Kruchten's article **From Waterfall to Iterative Development: A tough transition for project managers** (Rational Edge, December 2000) and **Going**

We use cookies to ensure you get the best experience on our website.

- Per Kroll's **Transitioning from Waterfall to Iterative Development** (Rational Edge, April 2004)
- Joe Marasco's **The Software Development Edge: Essays on Managing Successful Projects**
- Kurt Bittner's **What is Iterative Development? Part III: The Management Perspective** (also see **Part II: the Customer Perspective**, and **Part I: The Developer Perspective**), from the Rational Edge, Mar-May 2005
- Effective Governance practices for iterative software development, Mark Lines (Rational Edge, Feb 2005)
- **Iterative Software project planning and tracking**, Johanna Rothman
- Heuristics for Iterative Software Development, by Drako Sotirovski, IEEE Software May/June 2001, Involving Customers Early and Often in a Software Development Project, Laura Rose, Rational Edge, January 2006
- Iterative Software Development - a Practical View, by Morten Korsaa et.al., from DF-16 (Datateknisk Forum), Denmark 2001
- A Report of Development Lifecycle Methodologies Compared
- Controlling Iterative Software Development projects - The Challenge of Stakeholder & Technical Integration, by Bendik Bygstad, ICSS2004 proceedings
- COCOMO-based effort estimation for iterative and incremental software development, Oddur Benediktsson et.al., Software Quality Journal, Nov 2003

Acknowledgements:
Many thanks to Frank Schophuizen, Craig Larman, Ron Jeffries, Stephen Gordon, Scott Ambler, Mary & Tom Poppendieck, Jeff Sutherland, Paul Oldfield, Geoffrey Slinker, and Scott Preece provided lots of useful feedback, suggestions, and resources (as did numerous other contributors on the various "agile-related" YahooGroups forums).

Topics: change management, lifecycle models,

We use cookies to ensure you get the best experience on our website.

ABOUT THE AUTHOR

Steve Berczuk

Steve Berczuk is a Principal Software Engineer with experience as a manager, Scrum Master and technical lead in Boston, MA. The author of *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, he is a recognized expert in software configuration management and agile software development. Steve is passionate about helping teams work effectively to produce quality software. He has an M.S. in operations research from Stanford University and an S.B. in Electrical Engineering from MIT, and is a Certified ScrumMaster. Contact Steve at steve@berczuk.com or visit berczuk.com and follow his blog at blog.berczuk.com.

Brad Appleton

Brad Appleton is an Enterprise Agile+DevOps leader, coach & manager, and seasoned DevOps/ALM/CM solution architect at a large Fortune 100 company. Currently he helps organizations and teams scale, adopt and apply lean/agile development methods and DevOps/ALM/CM practices and tools. He is co-author of *Software Configuration Management Patterns*, the Streamed-Lines branching patterns, and a former section editor for *The C++ Report*. You can read Brad's blog at blog.bradapp.net.

Robert Cowham

Robert Cowham has long been interested in software configuration management while retaining the attitude of a generalist with experience and skills in many aspects of software development. A regular presenter at conferences, he authored the Agile SCM column within the CM Journal together with Brad Appleton and Steve Berczuk. His day job is as Services Director for Square Mile Systems whose main focus is on skills and techniques for infrastructure configuration management and DCIM (Data Center Infrastructure Management) - applying configuration management principles to hardware documentation and implementation as well as mapping ITIL services to the underlying layers.

UPCOMING WEBINAR

We use cookies to ensure you get the best experience on our website.

LETS HANG!

FEATURED RESOURCES

SECURITY
Shifting Security Left: DevSecOps eGuide
White Paper | StickyMinds

Log In or Register to post comments

USER COMMENTS

0 comments

YOU MAY ALSO LIKE

SECURITY
Agentic AI Opens Up New Security Flaws
Nahla Davies | August 27, 2025

We use cookies to ensure you get the best experience on our website.

SECURITY

Countering the Threat of Ransomware-as-a-Service (RaaS)

Nahla Davies | August 22, 2025

SECURITY

Building Trust in Blockchain: The Role of Smart Contract Testing

Ekaterina Egorova | November 18, 2024

Join the crew for FREE! Access exclusive content, contribute, and more.

See All Member Benefits

We use cookies to ensure you get the best experience on our website.