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

STICKYMINDS

Agile Testing DevOps Security AI

Requirements-Based Development: A Software Configuration Management View ARTICLE

By Steve Berczuk , Brad Appleton Robert Cowham | November 30, 2005

SUMMARY

It seems so obvious that we should develop systems based on requirements, and yet it turns out to be rather hard to do and thus many organizations are doing it very badly. From a software configuration management standpoint, we could perhaps leave the whole process of requirements engineering to one side and focus on the management of requirements and thus the aspects of change control and traceability. That would perhaps be unduly ducking the issue, and, of course, we can't resist giving an opinion anyway!

For definitions and descriptions of the process of gathering requirements, you can do a lot worse than use SWEBOK as a starter [1]. Requirements can be split into: functional, non-functional (interface, quality and other "ilities," performance, safety), derived, behavioral and operational. Dennis Linscomb [2] makes some interesting recommendations as to how the requirements aspect of CMMI might be improved to reflect a level of maturity in requirements.

So first a quick review followed by an agile SCM slant on things and some suggestions for improvement.

Big Bang (Waterfall) Developments

What came to be known as the waterfall approach presented a vision of a set of requirements all specified upfront which could be verified, signed off and then implemented with an ecstatic set of users at the end of it. This turned out to be a siren call which resulted in the shipwreck of many, many projects over the years. It is very hard to specify requirements, since often no one knows precisely what they want, and in any case the requirements are almost certain to have changed by the time they are implemented. Interestingly, there are situations where a rather accurate requirements definition is available—for example re-engineering or re-developing an existing system. In such cases the requirements process can be to point at the existing system and

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

Read More

Of course not all waterfall projects were failures. One of the authors well remembers a 7 figure fixed price development project where the requirements specification mixed one liners that implied whole subsystems and months of effort and in other parts spent pages on specific details of how certain screens should look (that turned out not to be needed at all). The fact that the project was a success was down to some excellent people being involved on all sides, and also that while we didn't quite ignore the spec, we did rather rewrite it. The project was likely to have been even more successful if done in a more evolutionary manner since the implications of various (some relatively minor) decisions did not fully manifest themselves until it was much too expensive to change them.

Evolutionary Development

This approach has been around for a long time, based on the idea that the faster you can deliver something, the more feedback you get and the more likely you are to get something useful. Richard Brooksby, in "Changing How You Change" [3] discusses the process of developing software to add value to your company, e.g. by understanding what is valuable to customers and solving their problems. He nicely summarises Tom Gilb's work in this area. Richard came up with the wonderful image of seemingly random changes in the software that often occur—very much like the apparently random (Brownian) motion of small dust particles in a gas. Have you ever seen new release of software come out with apparently pointless features and changes? The sign of a company that doesn't know which features are valuable, or that doesn't know how to ensure that they are the ones that get delivered.

The Agile Approach

For Agile methods, the evolutionary approach is fundamental—without it there is no agility! It provides closer conformance between delivered features and actual needs, and shorter lead time for new features. As Martin Fowler writes [4] "The power of shipped, running code is enormous. It focuses customer attention, grows credibility, and is a massive source of learning." This last point is interesting: we referred last month to Phil Armour's book **The Laws of Software Process**, where he maintains that software is not a "product" in the usual production-oriented sense of the word, but that software is really a medium for capturing executable knowledge. There has been a school of thought that accurate requirements could lead to detailed modeling and then on to a full system implementation with most of the process automated. Dave Thomas in "MDA: Revenge of the Modelers or UML Utopia?" [5] writes: *executable specification* is an oxymoron—if the specification were truly executable, it would actually be "the thing." Otherwise, it would merely model "the thing," which is by definition partial and incomplete.**The Lean Approach**

The lean approach has a slightly different viewpoint. In an attention grabbing presentation by Mary Poppendieck of Lean Software Development fame [6], she made the point that you make more money by increasing your ratio of outputs to inputs, and a key way to improve your productivity is just to do less work! Provide only what the customer will pay for. A quote from Jim Johnson of the Standish Group: users typically

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

Where do all these extra features come from?

- We ask the customer what they want
- We reward them for thinking of everything ("scope")
- We penalise them for adding features later ("scope creep")

So we effectively train them to go for a humungous, all singing all dancing set of requirements up front! This reinforces requirements as a key differentiating practice. For some alternative ideas she recommends Minimum Marketable Feature Set (Mark Denne & Jane Cleland-Huang, ***Software by Numbers; Low-Risk, High Return Development***, Prentice Hall, 2004).

The Power of Scenarios

Just a quick mention for a very powerful technique—that of writing scenarios to tease out requirements and desired functionality. If you pitch these correctly they can provide an excellent way of communicating the needs of the users. For some examples, see Joel Spolsky's examples [9]. The related technique that has sprung to the fore is use cases as describe about by Alistair Cockburn [10] in his two books and other related papers.

Managing Requirements

So let's get a little more practical and look at some of the details of managing requirements. From an SCM point of view this means treating them as configuration items and then implementing the classic processes of: identification, change control, status accounting and configuration audit.

Requirements Tools

To date, the most widely used requirements management tool is Microsoft Word (followed closely by Excel). The huge advantage is the extreme flexibility of these documents as a format for capturing and recording requirements. Word is very convenient and efficient for producing documents (including the majority of these articles!) and its universal availability is another major factor.

The biggest problem with this is that Word documents are frequently managed badly as configuration items. Problems include:

- Version identification and control for baselines—which version is valid for which release?
- Binary format makes comparisons difficult and thus tracking changes between versions of the documents difficult

The recent announcements that the next version of MS Office will default to XML formats sounds interesting in that it should make version control much easier.If you store documents in your repository with the code this makes it easier to track them and do

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

tools). It is possible to compare versions of Word documents using built in comparison facilities (we don't recommend using the built in versioning though). Manual merging is often surprisingly easy if you have a couple of windows open with differences highlighted in each and some judicious copy and pasting gets the job done.

Of course the specialist requirements tools claim to solve a lot of these problem, although usually at a not insignificant cost. In our January crystal ball gazing column [7] we referred briefly to some of the advances we expect to see in requirements tools: they typically have wonderful facilities for attaching and viewing meta-data and links and taking a container-based view of collections of items, while lacking considerably in areas of branching and merging and baselining when it comes to having to support multiple projects/variants/sites. The Source Code version control tools have the opposite problem. We expect to see considerable improvement in this area, and unified repositories potentially offer big advantages. We have written before [8] about the use of story cards within agile methods such as XP, and our recommendation to balance the use of physical cards with appropriate electronic backup.

Task-Based Development (TBD)

This implies that changes to software should be associated with specific tasks—this enables traceability. If you take this approach to its logical conclusion and mandate that no change can take place without an associated task you run the risk of people working around the problem. They associate a change with any old convenient task, and the traceability becomes much less useful. Especially for agile development you need to allow for activities such as refactoring. The benefit of this is not immediate, and yet if it isn't done regularly (mercilessly in some circles!), cruft builds up which slows the whole tempo of development. How do you know when enough refactoring has been done—that is usually a judgment call, and should be left to your developers, especially the senior ones—after all, why else do you employ them?! Done well, TBD provides a huge amount of benefit for very little overhead.

Automate, Automate, Automate!

However, we have seen any number of organizations where the actual linking of the changes checked in to the requirements involves various manual steps—copying and pasting from one application to another, or retyping information. Little steps, and some frequent slips—what seems a small overhead can mount up very quickly. Even if your tool doesn't directly support this, some judicious scripting works wonders.

One of the authors had a recent experience reviewing a team of over 100 developers. They had some review processes in place which required them to package up diffs of changes and get them reviewed. Someone had written a simple script to help automate part of this process. The script did diffs of all checked out files, not just the ones specific to that task. This required manual editing of the output file before attaching to an email. By adding 3 characters to the script we solved that problem saving a few minutes a day each for 100+ people—mounts up, doesn't it! For ideas and the whole ethos behind automation, see the work of the **Pragmatic Programmers**.

Testing Requirements

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

frequently improved (simplified) code—code that is easily testable is usually easier to understand than code that isn't. Quite a bit of work has been going in to acceptance test frameworks which are a little more difficult. Simple table driven specifications of tests such as those used in **Fitnesse**, can not only provide security but also help refine requirements since they provide a language to talk to the end user in that can be much more precise about the meaning of requirements. There are many challenges, particularly in the areas of GUI testing, but this is a very valuable practice to investigate.

Applying Agile Principles to Requirements

A good requirements process doesn't happen by magic, and certainly isn't produced just by tools. It requires hard graft and intelligence, both in the discovering of requirements and in their management and integration into the whole development process. Some principles which are helpful:

- DRY (Don't Repeat Yourself)—how many times is information copied backwards and forward between systems and applications? How can you minimise this, or automate it?
- YAGNI (You Aren't Gonna Need It)—evolutionary development and prioritizing of requirements for the upcoming delivery, postponing all sorts of details about other require
- Test First Development—get feedback on your requirements by working on acceptance test frameworks

Conclusion

A good requirements process is fundamental to software—heading a few steps in the right direction is usually better than zooming off ever so efficiently in totally the wrong direction! Being bad at discovering requirements is a quick way to failure. Of course even with excellent requirements you still have to develop the appropriate software. There may not be a magic bullet, but you can evolve from smooth bore muskets to sniper rifles through applying good software engineering and agile principles, and of course sound SCM practices too!

References and Additional Resources

[1] Software Engineering Body of Knowledge, **www.swebok.org**

[2] Requirements Engineering Maturity in the CMMI, Dennis Linscomb, Crosstalk

[3] Requirements and Change, Richard Brooksby, **http://www.ravenbrook.com/doc/2003/02/24/requirements-and-change/**

[4] Is Design Dead? Martin Fowler, **http://martinfowler.com/articles/designDead.html**

[5] "MDA: Panacea of the Modelers and UML Utopia?" by Dave Thomas

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

June 2001 (Vol. 9, No. 5 and 6—see full article at
http://www.poppendieck.com/lean.htm)

[7] Future of Agile CM, CM Journal, Jan 2005 (Vol 4. No. 1) by Brad Appleton, Robert Cowham and Steve Berczuk

[8] ***The Agile Difference for SCM***, CM Journal, Oct 2004 (Vol 3. No. 10) by Brad Appleton, Robert Cowham and Steve Berczuk

[9] User Interface Design for Programmers by By Joel Spolsky,
http://www.joelonsoftware.com/uibook/chapters/fog0000000065.html

[10] Writing Effective Use Cases, by Alistair Cockburn,
http://alistair.cockburn.us/usecases/usecases.html

Topics: configuration management, requirements, requirements management,

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.