CM Crossroads

the Configuration Management Community

Release Management – Making it Lean and Agile by Robert Cowham

A collaboration for Agile SCM by **Robert Cowham** and **Brad Appleton** – August 2004

This month we have a guest author, Robert Cowham, to write about this month's theme of Release Management, and how to make it more agile by borrowing principles, patterns, and practices of Lean and Agile Development (in addition to some tried and true SCM practices of course).

Release Management

Release Management is an awesome responsibility that plays a vital role in the success of a software development project (see [1] and [2]). Releasing is often considered to be an activity that happens near the end of the process - a necessary evil perhaps, but no more. This article contains some thoughts on the impact that the various activities which form part of software development, delivery and maintenance have on releasing.

What can make our releasing easier? Perhaps not surprisingly, quite a bit - the challenge is to influence the appropriate parts of the process.

I hope to provoke some thought and give some pointers to more detailed explanations of individual ideas and references.

What is a Release?

My definition is getting a version of "the system" into the hands of the users, where the system may be one or more executables,
web pages or web sites, associated documentation and help, related backend or third party systems, associated support systems, embedded systems, etc.

The system needs to be something that works and that performs valuable functionality for those users. A release of the system "on the shelf" is about as much use as a chocolate teapot.

For many internal systems, organisations would benefit tremendously from treating it as a product development for external use, which
typically have higher standards applied to them.

Common Release Problems

From the view point of the business, releases frequently:

- **take too long** - a new release which misses the market opportunity may miss millions in potential revenue.
- **are unpredictable** - one of the best diagrams I saw about the advantages of CMM showed the predictability of project schedules
  coming down from an average variance of 30 weeks to a variance of less than 4 weeks over a 2

year period as the organisation moved to level 2. One example I have come across is Symbian. They produce the operating system to go in millions of mobile handsets. Software release dates are planned in to schedules including factories making those handsets. The cost of unpredictable or missed schedules is potentially massive.

- **have poor quality** - systems with too many support calls and too many bugs. The impact on public trust caused by bad releases can have a very long-term effect. Look at Microsoft's efforts with "*Trustworthy Computing*" [3] and other related efforts to improve public perception of some of their products (though other factors are of course also present in this case).

- **are not useful** - does the system actually support useful functionality needed by the business, or is it mainly technical bells and whistles?

The business typically wants to manage risk sensibly andgain the maximum advantage from releases.

There are many other problems with releases, such as:

- **contents of the release not being planned** - the contents are produced in a rather ad hoc manner with all code finished by a certain date integrated in some fashion and shoved out the door.

- **contents unknown and uncontrolled** - I have seen many systems where people did not know exactly what had gone in to particular releases. I know of one company producing a fairly widely sold tool where they had very loose "controls" on releases. They didn't know what had gone out the door to various customers, and which variants were in the field. Myriad versions lay around which might or might not have been released. Support and reliability were a nightmare.

- **distribution/installation** - frequently overlooked areas, or at least overlooked until too late. A system that can't be got easily into the hands of users and installed on their system is not terribly useful. Upgrade/downgrade are also vital considerations.

Making Releases Easier

So let's look at what we can do to make releasing easier.

Planning

As with most things, the better we plan, the better the end result is likely to be. So what activities do we need to plan? Well if we consider the typical lifecycle of software products, it contains activities such as:

- Identifying the Business case and gathering requirements (market/features)

- Architecture/Design

- Detailed Design, Code and Test

- Integration

- Deliver/Install
- Maintain/upgrade

Perhaps not surprisingly, all of these activities can have an effect on how easy it is to release a product or system.

Lean Software Development Principles

This is a fascinating book by Mary & Tom Poppendieck which I heartily recommend [4]. I also attended a keynote talk by them a few
months ago in which they talked about how to "*Make More Money*" [5] - an attention grabbing title! You do this by increasing the ratio of outputs to inputs, and a key recommendation was to **do less work**.

She mentioned Jim Johnson (also noted Martin Fowler's summary of the XP2002 conference [6]) on the large proportion of features that aren't used in a software product. He quoted two studies: a DuPont study quoted only 25% of a system's features were really needed. A Standish study found that 45% of features were never used and only 20% of features were used often or always. 65% of features in systems are rarely or never used. A system that has fewer features is likely to be easier to release!

Mary referred to the term "*Minimum Marketable Feature Sets*" in the book "**Software by Numbers; Low-Risk, High
Return Development**" by Mark Denne & Jane Cleland-Huang [7], as a way to conduct your requirements gathering (along with a proposal, theory and metrics for an *incremental funding method - IFM*).

Some other concepts from **Lean Software Development** that are of interest, particularly from the principles of lean thinking [8] and
rules of lean programming [9]:

- Streamline your process using value stream mapping - if you track a change request or similar through your development system and record the amount of time it is being worked on vs. the amount of time it is waiting for something to happen, you will typically find most time is spent waiting. For example, waiting for customer reviews can often take weeks to be scheduled. Processes like Agile methods look to minimise this "down time".
- Delay commitment on decisions
- Shorten the customer feedback loop and deliver fast
- Requirements change so you need to manage this.

See Appleton's presentation on "Agile Configuration Management Environments" [10] for some ideas on how some of these
principles and rules can be applied to configuration and release management

Architecture

The architecture of a system is often already defined as part of the business case, but it can have a tremendous affect on the ease of release. Thus the tremendous growth in browser based applications where the only thing that needs to be upgraded is the server. This is in contrast to the greater usability that is still achievable with thicker client applications.

Of course Microsoft recognised the major problems that organisations had with installing fat client applications and versioning
conflicts inherent in "DLL Hell". Their approach has been to produce the .NET framework with its "xcopy deployment" architecture.

One of the problems in deciding on the architecture for a windows based application at the moment is that developing with the .NET framework requires a 20Mb download for many end users. This requirement alone may negate all the development advantages that .NET brings in comparison to COM based solutions since it reduces the potential market. When are the majority of users going to have the .NET runtime installed, or is the broadband penetration going to be such as to make this less of an issue? (Of course it still won't go away since there will be several versions of the .NET framework out there - you may need to develop for the lowest common denominator).

There was an interesting piece recently in the June issue of *Joel on Software* [11] talking about extending HTML to make web
applications work better, and Joel's piece on *How Microsoft Lost the API War* [12] suggesting that making DHTML work better in Internet Explorer was too dangerous to Microsoft's core business as it would make servers more of a commodity! Worth a consideration.

For an internal product where you have more control of the desktop software and capabilities, such decisions can be much easier - choose the architecture that makes development easier.

The concept of application services provision by companies such as Salesforce.com can also make life a lot easier for customers (at the
risk of entrusting your data to a third party). As an ASP, producing new releases is vastly easier with full control of all the servers and the data.

System Environment

The environment in which a system functions is very important. This includes many items such as databases, third party systems, specialised hardware etc. Not only do you have to consider compatibility between releases of the system, but testing can be very difficult. Resources to fully duplicate live environments can sometimes be prohibitive, requiring more risky testing procedures. Anything that can be done to minimise the dependencies on such external items is going to make releasing easier.

Databases are a common challenge for controlling using CM tools. You need to identify structure changes, upgraded/downgrade scripts, etc. Enough for a whole article on its own!

Process/Patterns

There are many applicable patterns for the process of development which can ease releasing of software. It has long been understood
that software design principles such as modularity, cohesion, coupling, understandability and adaptability are all of benefit, and these carry through to the releasing of software.

These days, the production of software as groups of components is very widespread since it builds on these principles, and makes the construction of larger systems from component parts more manageable. I liked the metaphor mentioned by Laura Wingerd of Perforce Software at the BCS CMSG 2003 Conference [13]: it is easier to take a group of 40 adults on a trip than 5 children. The adults can be treated as components each of whom is responsible for their own luggage etc. For the children, you have to ensure they have packed appropriately down to the last toothbrush.

SCM Patterns

There are a number of SCM Patterns particularly applicable to releasing. From the book of the same name by Berczuk & Appleton [14], I
would like to briefly mention:

- Task-Level Commit
- Integration Build
- Release Build [15]
- Smoke Test/Regression Test
- Release-Prep Codeline
- Mainline
- Codeline Policy

For more details, please see the book and previous articles in CMCrossroads.com "Agile SCM" column (particularly [15],[16] and [17])

Producing Release Notes

From my personal experience of managing releases, I have found one activity to be a key indicator for process health - the production of Release Notes.

These are typically a list of changes: bugs fixed and features added. They are very useful to everyone to understand what is in the release, how it differs from previous releases, and whether a particular problem will be fixed by upgrading or not.

I have certainly had the experience of producing release notes being a major chore. Having volunteered (or been volunteered) to do the job, I had to find out what had changed. The first approach was email all the developers asking for a list of changes. One deafening silence later, and I would go around and bug them in person until they gave me some sort of list. This I would use to compare with and revise the project manager's rather optimistic list of changes. Being a naturally suspicious person, I would typically grub around in the code looking at differences between the release version and the previous version and trying to match changes with the official list. This would frequently turn up extras, or omissions. All in all, a rather painful activity, especially as it was usually left until near the deadline

when things were being madly rushed out the door.

Life is much easier if *task-based development* [18] is followed.

- Firstly the release contents are planned - the list of open bugs and planned new features is tracked and they are assigned to developers. No code changes are made without associated tasks (this doesn't need to be enforced by a tool though it needs to be made easy to find out, since appropriate carrot and stick techniques work wonders).
- Then you need some simple workflow to track code fixes and whether they have been coded, tested and verified. The linking of tasks to code changes (or change sets ideally) should be made as easy as possible.
- Finally, it is helpful to have good tool support to track associations across release lines. See the discussion by Gareth Rees in "Can we ship yet?" [19].

Practice makes perfect...

Not surprisingly, a key way to release better is to do it more frequently! Frequent releases highlight the problems, and force you to address them. Manual steps or painful processes that are possible once or twice a year, become unmanageable on a more frequent basis. This alone focuses the mind and leads to improvement.

Agile Development lends itself to this with the following practices (see [20] and [21]):

- Small, Frequent releases (regular cycle)
- Release Plan
- Iteration Plan
- Continuous Integration
- Test-driven development
- Unit testing
- Acceptance testing
- Code should always work!

Don't forget to balance these practices against your current project by taking into consideration your organization's "culture" and project's context. Boehm and Turner's "**Balancing Agility and Discipline**" [22] and Larman's "**Agile and Iterative Development: A Manager's Guide**" [23] can prove helpful in doing this.

Automation is a basic requirement for frequent releases. Automation of unit testing in agile methods leads to much greater confidence that the system works and that bugs have not been introduced. Automated builds catch integration problems early making them much easier to fix. Automation of the production of release notes as discussed above makes the project much more controlled and less subject to surprise. Automation of installs and upgrades are also key requirements for systems these days.

A frequently underestimated part of Agile methodologies, is that of "*information radiators*" - a central whiteboard or similar for the team where current status and progress are shown in an easily visible manner. This makes it obvious what needs to be done, and who is currently doing what.

See "**The Pragmatic Programmer**" by Hunt and Thomas [24] for more recommendations on automation (including their new book on Project Automation [25]).

Also, the BCS has an event on the 12th October 2004 addressing Agile methods and CM (see http://www.bcs-cmsg.org.uk).

Case Study

One company I have been involved with over the last 2-3 years produces a large treasury system used to trade billions on a daily basis. When I first started working with them, they had 15 clients spread over 3 versions of their system (which had many, many configuration possibilities). The system had a VB front end and a 4GL backend. The interesting thing was that they had a single development workspace to work on all 15 releases of the system. This meant that a customer using an older release who wanted a single bug fix was forced to take the latest version of the system (with the bug fix).

The latest version had seldom been tested against that customer's configuration parameters, and it usually took weeks for the new release to stabilise.

Like most such processes, this had grown in a higgledy-piggledy fashion over the years in to the unholy mess it was then in. No big bang fixes were possible and it has been a slow and steady process of enhancements that have taken them to a much more controlled environment today. Migrating to a single SCM tool across all platforms was an early step. Separate build and test environments for testing brought major advantages. Branching to support older releases was also beneficial. It's still a work in progress!

Conclusion

Making a system easy to release is very much a holistic “thing” - you need to look at the whole lifecycle of the development. Plan it, up front and throughout; adjust your processes and use appropriate patterns. Practice releasing regularly, and even if iterative and agile methodologies are not possible in your particular organisation, steal as many practices as appropriate and apply them.

If you are assigned to look after releases in your organisation, start working back through the whole process looking for places that you can influence things to make your job easier. Bad decisions early in the process can make releasing a nightmare - don't accept them. Identify the problems, point out the ramifications, and work with the appropriate people to improve those areas.

Happy Releasing!

References

[1] **Software Release Methodology**; by Michael E. Bays; Prentice-Hall PTR 1999

[2] *Release Management,The Super Discipline*; by Mario E. Moreira; CM Crossroads Journal, November 2002 (Vol. 1, No. 11)

[3] *Trustworthy Computing*; by Craig Mundie; Microsoft Whitepaper, October 2003 (see http://www.microsoft.com/mscorp/innovation/twc/twc_whitepaper.asp)

[4] **Lean Software Development: An Agile Toolkit**; by Mary and Tom Poppendieck; Addison-Wesley 2003 (see http://www.poppendieck.com)

[5] *"Software Development Productivity / Make More Money!"*; presentation by Mary Poppendieck (see http://www.poppendieck.com/pdfs/Productivity.pdf)

[6] *The XP2002 Conference*; summary write-up by Martin Fowler at <http://www.martinfowler.com/articles/xp2002.html#N100FC>

[7] **Software by Numbers – Low Risk, High Return Development**; by Mark Denne and Jane Cleland-Huang; Prentice Hall PTR, 2004 (see http://softwarebynumbers.org)

[8] *Principles of Lean Thinking*; by Mary Poppendieck; **2002 Conference on Object-Oriented Programming Systems, Languages, and Appplications** (OOPSLA 2002); Seattle, WA, November 2002; (see http://www.poppendieck.com/papers/LeanThinking.pdf)

[9] *Lean Programming*; by Mary Poppendieck; Software Development Magazine, May-June 2001 (Vol. 9, No. 5 and 6 -- see full article at http://www.poppendieck.com/lean.htm)

[10] *Agile Configuration Management Environments*; by Brad Appleton;Chicago Software Process Improvement Network (CSPIN) presentation, March 2004 (see http://acme.bradapp.net/#AgileScm)

[11] *Joel on Software - June 2004* comments (see http://www.joelonsoftware.com/items/2004/06/17.html)

[12] *How Microsoft Lost the API War*; Joel Spolsky, from *Joel on Software - June 2004* (see http://www.joelonsoftware.com/articles/APIWar.html)

[13]*Container-based SCM and Inter-File Branching* by Laura Wingerd; **BCS CMSG 2003 Conference** (see http://www.bcs-cmsg.org.uk/conference/2003/abstracts.shtml#Wingerd)

[14] **Software Configuration Management Patterns: Effective Teamwork, Practical Integration**; by Stephen P. Berczuk and Brad Appleton; Addison-Wesley, November 2002

[15] *Build Management for an Agile Team* by Steve Konieczka, et.al.; CM Crossroads Journal, October 2003 (Vol. 2, No. 10)

[16] *Codeline Merging and Locking: Continuous Updates and Two-Phased Commits*, by Brad

Appleton, Steve Konieczka and Steve Berczuk; CM Crossroads Journal, November 2003 (Vol. 2, No. 11)

[17] *Agile Change Management: From First Principles to Best Practices*; by Brad Appleton, Steve Berczuk and Steve Konieczka; CM Crossroads Journal, August 2003 (Vol. 2, No. 8)

[18]*SCM Patterns: Building on 'Task-Level Commit'*; by Austin Hastings; CM Crossroads Journal, June 2004 (Vol 3. No. 6)

[19] *Can We Ship Yet*?; by Gareth Rees; **2001 Perforce User's Conference** (see http://www.perforce.com/perforce/conf2001/rees/WPRees.html)

[20] **Extreme Programming Explained: Embrace Change**; by Kent Beck; Addison-Wesley, 2000

[21] **Planning Extreme Programming**; by Kent Beck and Martin Fowler; Addison-Wesley, 2001

[22] **Balancing Agility and Discipline: A Guide for the Perplexed**; by Barry Boehm and Richard Turner; Addison-Wesley 2003

[23] **Agile and Iterative Development: A Manager's Guide**; by Craig Larman; Addison-Wesley, 2003

[24] **The Pragmatic Programmer: from journeyman to master**; by Andrew Hunt and David Thomas; Addison-Wesley, 2000 (see http://www.pragmaticprogrammer.com/)

[25] **Pragmatic Project Automation**; by Mike Clark; The Pragmatic Bookshelf, July 2004 (see http://www.pragmaticprogrammer.com/starter_kit/auto/)

Robert Cowham *is a Principal Consultant at Vaccaperna Systems Ltd in London, UK. He mainly provides SCM consultancy and training, but still keeps his hand in development wise! You can contact him via*