Changing how you Change

Peter Jackson, Symbian Limited and Richard Brooksby, Ravenbrook Limited

Abstract

It's an interesting exercise to examine a company's configuration management needs and set up a system, but this paper discusses what happens a few years after that. Systems evolve to adapt to organisation change and customer requirements. How do you make sure that these changes are good ones and what do you do if you realise that things are going wrong? This paper is a case study of just such a situation, showing how it came about, what we are planning to do about it and how similar situations might be prevented in other organizations.

1. Introduction

This paper is about a big change to Symbian's configuration management. We describe the evolution of Symbian's configuration management process, how the current situation came about, what's wrong with it, and how a radical solution has been discovered.

The lessons we have learned should be of interest to anyone running a configuration management system supporting hundreds or thousands of developers.

2. Background

In A year with Perforce [PNJ2000], Symbian's transition to a new SCM tool combined with a new way of running its software production process was described. Symbian had moved from being a company used to supplying one customer at a time to a company supplying multiple customers with different device and UI profiles. Its previous production process needed radical change to keep up with this.

Symbian's product, Symbian OS™, is delivered in source code form to its customers, who are 'phone manufacturers like Nokia, Ericsson or Motorola. The source code is managed in a Perforce repository which today supports about 650 users. Symbian also works with a number of development partners whose work is integrated into the product in this repository. The source tree consists of about 100,000 files and last year, about 72,000 changelists were submitted.

The SCM tool and process changes were very successful and it is worth reflecting on the reasons for this and to try to abstract them to help with other change projects. They fall into two sets:

1. The prerequisites for change:
   1. The organisation is in pain;
   2. There is a solution available;
   3. Key people on the ground are able and willing to act as agents of change;
   4. Senior management support the change exercise.

2. The execution of the change:
   1. A lot of careful planning and thought goes into the change;
   2. this is coupled with extensive communication at all levels;
   3. the engineering community is fully involved in the change;
   4. everyone affected by the change receives training and documentation.

The organization must be ready for the change. If it is in discomfort then everyone is aware of the problem and is willing to look at solutions. Agents of change are the sort of people who can be
vociferous in their criticism of things in your company but will also enthusiastically adopt a vision and make it happen.

The importance of communication can't be overstated. It's essential that you frequently talk to as many of your staff as possible about what you're doing, because they will come up with reasons why the details of your plan won't work and suggest adjustments. People are always making decisions in the course of their work which you want to be consistent with your strategy rather than inadvertently interfering with it.

In addition, note that in any such exercise you have options of safe, incremental change or more risky but faster disruptive change. Our experience was that the disruptive change works well if everyone affected is involved and supports the plan.

What the *Year with Perforce* paper didn't cover is how you build on your successful change and keep your production system at optimum. You know that all systems change over time and you need to find ways of promoting good changes and correcting sub–optimal changes when they happen. The following sections in this paper describe how Symbian's production system evolved and how that evolution led to another significant change.

### 3. The rise of the release project

In 2000 then, Symbian's production system was centred around a submission to *mainline* process. This pattern of working is well described in *High-level Best Practices in Software Configuration Management* [Wingerd1998–07] (see [figure 1](#)). The mainline is a complete source configuration of the product, which was built and tested every day. Code would get into the mainline from development codelines. When development code was thought suitable for inclusion it went through a process to ensure that it met various standards, before being integrated into the mainline.

![Figure 1. Submission to mainline](#)

Submissions to the mainline were controlled by a team of *build co-ordinators* who were experienced practising engineers who carried out these extra duties on a rota basis. Submission policy was primarily about keeping the mainline in good working order.

To get the software to the customers the mainline was branched to release codelines when a defined set of functionality was complete. Release codelines were managed by *Release Projects* to ensure that they were of high enough quality to give to the customer. You can imagine a quality scale: in development quality is suspect, in the mainline we know what we're dealing with, and in release it was suitable for customers to build products with.
In principle, work on the release codelines would be confined to extensive testing and defect fixing but in reality it also meant implementing functional changes as the customers changed their ideas about what they wanted to be delivered.

Symbian's customers get components from all over the world, and Symbian OS is just one such component. If any of them are wrong or late then the production process is affected, and that's very expensive. There are state of the art factories full of robots poised to make devices, and it costs millions of dollars for them to be idle. There is more information about this larger production process in Co-operative development at Symbian [PNJ2001]. Symbian's customers were keen to make sure that deliveries were correct and on time.

The release projects at Symbian were therefore managed primarily to meet this requirement. They were very successful at this, which led to a demand for more functionality and significant growth in the Symbian organisation.

To illustrate that growth, here is a table showing the number of active engineers, measured by examining Perforce submissions.

<table>
<thead>
<tr>
<th>Year</th>
<th>Number of active engineers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999 (Aug–Dec)</td>
<td>201</td>
</tr>
<tr>
<td>2000</td>
<td>323</td>
</tr>
<tr>
<td>2001</td>
<td>388</td>
</tr>
<tr>
<td>2002</td>
<td>530</td>
</tr>
<tr>
<td>2003 (Jan–Feb)</td>
<td>328</td>
</tr>
</tbody>
</table>

Our data also shows that only 18% of the users in the 1999 list were submitting changes in 2002. This doesn't mean that the rest had all left the company, but it does illustrate how operational knowledge can be diluted as a company grows.

Alongside this growth, Symbian was working to manage itself in a more mature way, in order to continue to meet its customer's expectations. It used the Capability Maturity Model (CMM) [SEI1993–02] as a framework for this and was formally certified as operating at CMM level 2 in September 2001. A well–known side effect of this kind of progress is an increase in the level of bureaucracy in the organisation. This tends to peak as CMM level 2 is achieved and then diminishes again to more acceptable levels as the company learns how to tailor its processes. But while this effect lasts, processes tend to be harder to change.

Over time the senior build co–ordinators, whose function was perhaps considered a distraction by the release project managers, were replaced by less experienced staff and heavier reliance on a single standard mainline submission process. Mainline quality suffered a bit because of this and the problem was addressed by having the mainline placed under the control of the release projects. This was successful at bringing mainline quality back on track. Each release project tended to introduce its own policies and processes and these could be changed more quickly than the wider company standards.

However, these policies tended to be in the best interests of the current project and didn't always take wider issues into account. Furthermore, the release projects tended to be quite lengthy, so that there could be a considerable period passing between functionality being engineered and its final consolidation at the end of a release project. These lengthy projects meant that customer change requests were more likely to be a factor during the life of a project.
### 4. Overheads increase

Although the release projects were highly successful, by mid-2002 the company had noticed a worrying increase in effort spent on maintenance.

Symbian's original production model followed a standard approach in which release codelines were used to deliver planned functionality. The theory was that a release branch was derived from the mainline, and releases built from it from which the customers made products. To meet new customer requirements Symbian developed the mainline, and then made another release branch, and so on (see figure 2). The idea was that by the time Symbian was delivering version 3, customers should have lost interest in version 1, and should be on the way to using version 2. Unfortunately, this wasn't how they were behaving.

![Figure 2. Release branch theory](http://www.ravenbrook.com/doc/2003/03/06/changing-how-you-change/)

Old versions were refusing to die. The true picture looked more like figure 3. Why was that? The customers were comfortable with version 1 — they knew how to integrate it and make products out of it. They were less sure about version 2 and 3. Every time Symbian makes a change the customers have to do integrations into their own software. Those are risky and expensive. Instead, they were asking for functionality to be backported from later to earlier releases.

![Figure 3. Release branch practice](http://www.ravenbrook.com/doc/2003/03/06/changing-how-you-change/)

The V4 release project being branched from V3 was an obvious alarm signal to those experienced in configuration management. But a large company can take longer to react to these things. There is a
cognitive dissonance effect in which the company collectively tries to fit events into the framework they are used to.

So there was an increasing cost caused by having to maintain earlier versions of the system for longer than expected. The act of backporting also carries significant cost — engineers are having to work on the same functionality more than once. These costs coupled with the increasing intellectual discomfort about what was happening were the beginnings of sufficient organizational pain to take action. One action that was taken was to ask for a codeline strategy review.

This could be perceived as a tactical approach — maybe there was some way to organize the codelines to make backporting and defect fixing easier, and so reduce cost.

Tree diagrams (such as figure 2 and figure 3) mean different things to different people. They show the relationship between bodies of code, but to a CM specialist, codelines also define the policies and processes that surround how you make changes. Those are connected to the business process that you're running. Perhaps a more profound change was required. In responding to that idea one of the steps Symbian took was to engage an outside consultant, Richard Brooksby, to help us look at this with fresh eyes. A good thing about consultants is that they can move through an organization asking awkward questions and stepping on toes in a way that is hard for internal staff to do.

5. The master codeline

Richard spent many hours over a period of about a month investigating Symbian's situation. He compared Symbian's process with more standard models, and investigated the reasons for Symbian's special model. He came to some startling conclusions about what Symbian were doing [RB/2002-10-30]. He concluded that these release codelines weren't incremental versions of Symbian OS, but were in fact separate and incompatible platforms that the customers wanted developed and maintained all at once. They weren't evolutions of one specification, but were independent entities with their own evolving specifications. The notion of "later" and "earlier" releases was bogus. Symbian needed to embrace this fact. Furthermore, Symbian needed to turn the divergence of platforms into a convergence, or the proliferation would cause costs to increase without bound.

The way to do that is to put all the separate platforms into one master codeline (no small task), then deliver from that into version branches, which are produced at sufficiently short intervals so that the customer can make the transition from one to another without too much cost (see figure 4). Furthermore, introduce a rule which says that functionality can be only added along the mainline, and the only thing you're allowed to do on version branches is fix defects. So if a customer wants new functionality they have to advance along the mainline.
Peter Jackson reports that he nearly fell off his chair when he read the recommendation. This is the kind of "aha!" moment you sometimes get when you've had a problem for a long time, and you suddenly realize what you're facing. The realization is such a surprise because of the cognitive dissonance we mentioned earlier. The assumptions of your earlier model can dominate your thinking and obscure the truth.

Another important part of the proposal was to control the changes that went into the master codeline very carefully, to avoid a costly situation arising again. The proposal was to measure each change against customer requirements, and only allow it if it increased the value of the product [RB1999–05–20, section 6]. So what we needed to do was make sure that the submission policy to the masters was directly referring to customer requirements. The requirements include all of the customer's needs, not just the ones which used to drive those big release projects. One of these is a requirement which articulates the amount of change a customer can tolerate between versions. By measuring and enforcing that requirement, we can keep the customers up to date, and so keep control of the proliferation of configurations. This turns out to be a major change in business engineering in other parts of the organization (see figure 5). A lot of other changes are needed to produce an improvement in changes to code, with several feedback loops.
A third part of the proposal was to move to a more incremental development model. Development projects which take a long time to deliver result in large changes which can't be borne by the customers. Furthermore, large changes introduce large numbers of defects. Lastly, the value of a long project isn't realized until the end. Often, the requirements change, diluting the value, so the change is worth less by the time you get it out. It's much less risky to develop incremental improvements, each of which has independent value, even if the total amount of work required appears greater. In fact, it's likely that the requirements will shift and the later stages will be different than you imagined anyway. You can't afford long changes in a rapidly changing market.

In Perforce terms, an important result of this is that there should be no long-lived development codelines, which have the tendency to diverge from the master codeline. Instead, each incremental piece of development is based on a branch of high quality master sources (see figure 6).

![Figure 5. The business context](image)

**Figure 5. The business context**

**Figure 6. Incremental development branches.**

6. History repeats itself: our conclusions

So once again Symbian is making fairly major changes to its organization. We're faced with the same dilemmas as before: softly softly, trying not to upset the production process, or going for a bigger bang and making some dramatic changes all at once in the hope of minimizing the total disruption. History is repeating itself. We suggest incremental changes, but the feedback we get from across the organisation is that they like what we're doing and they'd prefer to jump into it sooner rather than later. It'll be interesting to see whether this larger organization can accommodate this change as well as the CMM level 1 organization back in 1999 was able to.

But, talking of history repeating itself, what's to stop it repeating itself in another way? What's to stop us once again drifting away from our ideals and ending up with a situation which we have to change again? We believe the answer is to institutionalize the key principles behind how you organize your business, so that you have a change process that will take them into account, even though the original people have moved on. To do that not only must you have good processes for changing processes, but you must also must make sure your staff are fully trained in what it is you're doing and the reasons you're doing it. So don't lose sight of the training and communication activities that you need to sustain in order to maintain that corporate intelligence.

You should also make sure you're measuring what you care about. If there's some aspect of your software production process that's critical to its success then you need to make sure you can quantify that and publish that information. Make sure your senior managers and whole organization is aware of it, so you can recognize deviation when it occurs and not 6 months later.

You should make sure the organization understands what's valuable by continuously developing and maintaining a set of requirements [RBI1999–05–20, section 3]. These requirements must include every aspect of the product that affects its value: not just a technical list of desired features, or even a
specification of technical attributes like performance and size, but attributes such as the customer's acceptable cost of integration, the effort required to learn about the product, the product's reliability, and so on. (There will be more on this topic in Richard's paper, Requirements and Change [RB\textbackslash{}2003-02-24].) All work on the product must be driven by, justified by, and measured against these requirements. The configuration management process has to support this. Make sure you understand how the requirements constrain the kind of changes you can make. Make sure your configuration management process fits with these constraints. Make sure you review your configuration management process whenever the relevant requirements change.

Be aware that everything changes, and the system you set up today won't last forever, even if it's currently a perfect fit. Review your systems regularly, and make sure they are still meeting their requirements, just as you're always reviewing your products to see if they meet theirs.

Configuration management is about dealing with change, but it's easy to forget that the configuration management process itself must change.

A. References


\textbf{[RB\textbackslash{}1999–05–20]} "Product Quality through Change Management"; Richard Brooksby; Ravenbrook Limited; 1999–05–20; \texttt{<http://www.ravenbrook.com/doc/1999/05/20/pqtcm/>}. Richard's key white paper which is important background reading for many of the ideas connected with this work.


\textbf{[Wingerd\textbackslash{}1998–07]} "High–level Best Practices in Software Configuration Management"; Laura Wingerd, Christopher Seiwald; Perforce Software; 1998–07; \texttt{<http://www.perforce.com/perforce/bestpractices.html>}. Presented at the Eighth International Workshop on Software Configuration Management. This is a practical overview of important aspects of the production process. Some of this is the source of several principles now taken for granted at Symbian.

B. Document History

http://www.ravenbrook.com/doc/2003/03/06/changing-how-you-change/
C. Copyright and Licence

Copyright © 2003 Ravenbrook Limited and Symbian Limited. This document is provided "as is", without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this document. You may make and distribute verbatim copies of this document provided that you do not charge a fee for this document or for its distribution.

$Id: //info.ravenbrook.com/doc/2003/03/06/changing-how-you-change/index.html#18 $