PERFORCE SOFTWARE

2001 User Conference

Co-operative development at Symbian

Peter Jackson, software architect, Symbian Ltd.

Abstract

This paper describes how Symbian organises software development involving the use of third party suppliers and development partners. This includes a description of the organisational groups involved, how intellectual property is managed and the processes and patterns surrounding import, delivery and codeline management.

The author is a software architect working on software production processes at Symbian. A system manager and developer by background, he joined Psion in 1994 and became part of Symbian when it was formed in 1998. In 1999 he was part of the team responsible for introducing Perforce into the company and organising the accompanying process changes and training. Configuration management is now his key role.

Symbian

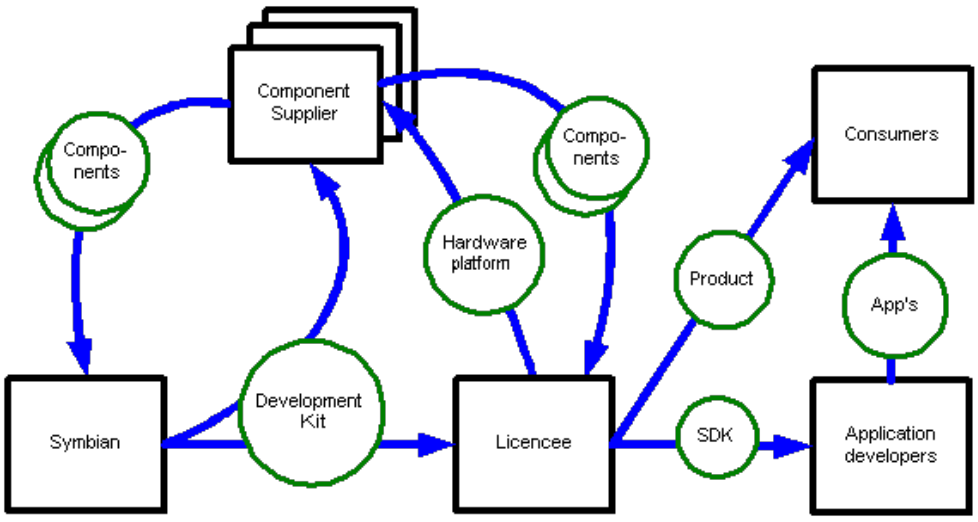
Symbian produces an operating system, called **Symbian OS™**, for **2.5G and 3G** 'phones. There are about 450 Perforce users. The server and development infrastructure is NT based and there are three major development sites. The main codeline contains about 78,000 files in 9000 directories.

This is a growth of over 50% since this time last year. The checkpoint size is approximately 10GB and the changelist rate is about 50,000 per year.

More background information about configuration management at Symbian can be found in ***A year with Perforce*** – a paper presented at the 2000 Perforce conference.

Symbian's product, described loosely, is a development kit that allows Symbian's licensees to build specific device products running Symbian OS. The development kit is also delivered to other software organisations who produce **components** that may be for Symbian OS licensee products or could be delivered back to Symbian for incorporation into the Symbian product.

Pictorially this might be represented as follows:

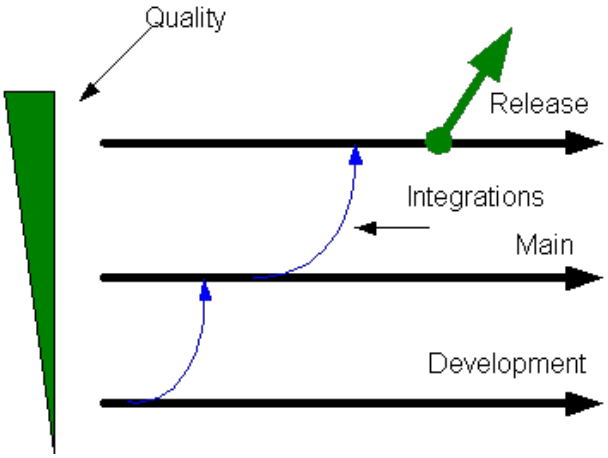


The management of software components is significant. Not all of the components that Symbian produces are available to Symbian's licensees in source form. Similarly some components that go into the licensee product are not available to Symbian. The end product is built from binaries and no one company has the ability to build from source all of the components that make up the end product.

The diagram also illustrates that Symbian's licensees build development kits for their own products which

are based on the Symbian development kit.

Internally, Symbian follows a development-main-release codeline pattern whose core is the mainline submission process that maintains a level of quality for leading-edge code. All product deliveries come from release codelines whose quality is more strictly regulated leading to the final release of a particular version of the Symbian product.



All development projects follow a life-cycle which reflects this progression of quality and each project has a configuration management plan that provides specific details on the codelines involved and their management.

Third parties

Symbian OS development has a great deal of third-party involvement which manifests itself in a number of ways:

1. Existing third party code with special licensing requirements;
2. Symbian code being developed in partnership with a third party;
3. Code being added to Symbian OS as a commercial delivery.

All of these situations exist in a typical Symbian OS release project. The development organisation has processes in place to deal with them and some of these processes are reflected in the codeline structures we use. We'll see more of this in a later section.

We have about 50 development partners now and are planning for this number to grow significantly in the near future as our development projects become richer in specialised functionality.

Intellectual Property

The necessary involvement of third parties in the development of Symbian OS means that it is difficult to cover the entire system under a single legal agreement. It is important that we clearly understand the legal requirements connected with every part of the code. To do this, we invented a set of 5 policy categories with which the code is labelled. Paraphrased, these are:

A. Confidential Source. Not distributed. All customisation performed by Symbian.
B. Third Party Confidential Source. Source for which Symbian's licence from the third party supplier forbids distribution in source form.
C. Jointly Developed Source. Not distributed at all except to the co-developer of the software or by special arrangement.
D. Reserved Rights Source. Provided to all licensees. Symbian retains the right to take back any changes to this source.
E. Standard Source. Source code which licensees are allowed to modify and include in products.
F. Example Source. Freely provided to all Symbian OS developers.
G. Application SDK Source. Code which licensees are allowed to modify and include in products. May also be shipped under licence as part of the licensees own SDK.

Category F and G code can generally be distributed without specific signed agreements. Our aim is to try and move as much code as possible into unrestricted categories.

Labelling is applied at directory level by adding a file called `DISTRIBUTION.POLICY` to the directory. Its

contents follow a trivial syntax which is enough to categorise the code in the directory and provide commentary if needed. All code in the mainline is labelled like this.

During construction of a software delivery, a tool is used to filter the code according to the entitlements of the recipient and to construct a report. Any directory not labelled is flagged as an exception and treated as category A.

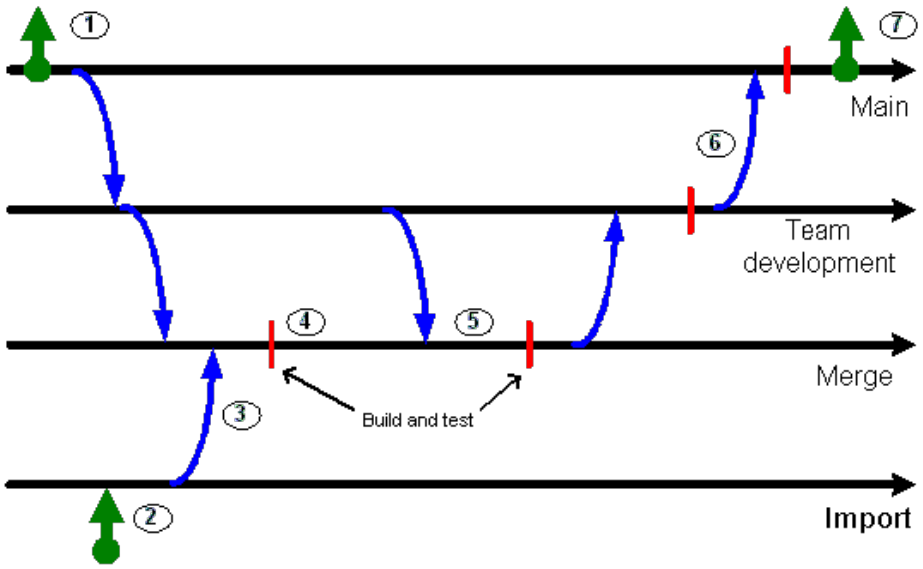
This mechanism provides the mechanical regulator for code distribution. Behind it lies a committee who approve all new code categorisation or changes to existing categorisation. There is also a partner management group to set up and manage contracts with third-party suppliers as well as a legal department.

Codeline patterns

Management of third party code during development is mostly concerned with extending the internal development cycle to include external engineers. A typical cycle would be:

1. Deliver initial baseline to the supplier, recording its configuration (changelist number and the depot specification of the codeline from which it was released);
2. Receive a new delivery of external code;
3. Integrate it with its baseline configuration (the configuration used by the third-party to develop the code);
4. Test the delivered code; (reject if necessary, just as we would reject an attempt by a local developer to submit defective code)
5. Integrate the code with the current configuration and test;
6. Integrate the tested changes into the mainline;
7. Deliver an updated baseline to the supplier.

The codeline diagram below illustrates this.



Normal development changes, extra **catchup integrations** and so on have been omitted for clarity. In this scenario, baseline code is delivered to the supplier from the mainline and code received from the supplier updates an import codeline using a detached client process (see *Perforce* ***technical note 002*** for details). The imported code is then merged with its baseline configuration in a merge codeline and tested in that context. Any new IPR policy files required are also added at this point.

Assuming the code is accepted, a catchup integration from the team development codeline is carried out and the new configuration is tested again and modified if necessary until it works.

The working code is then integrated into the team development codeline from which it makes its way to the mainline in due course. From here, further baseline deliveries can be made to the supplier.

In practice, there are several variations possible in this pattern, for example the point from which baseline deliveries to the supplier are made could vary.

The challenges in managing this process are to:

1. Maintain quality control across the entire code base;
2. Provide rapid feedback to the supplier so that they remain in step with the whole development;

3. Ensure syncronisation of code sent from a supplier with the delivery on which it is based.

Note that projects often have several suppliers and each would have their own import codeline. However, for sanity, a project would tend to make deliveries from one delivery codeline to all suppliers at the same time.

Another aspect of joint development is management of the development environment so that all developers are using similar environments and tools. In particular the tools to build and test the code should be standard. The development environment should be well defined (for example, compiler revisions) and documented. Any elements of a build script that are site-specific (for example, Perforce commands) should be segregated from the exportable part of the script.

Key roles

For the entire process to work smoothly and form part of a predictable delivery of the Symbian product, we must actively support a number of key roles in the development process. In the organisational structure we have:

Release management team
: Organises and runs regular builds and smoke tests of the mainline and active release codelines. Manages build co-ordinators.

Legal department
: Responsible for the correctness of all legal agreements and contracts.

Vendor management group
: Sets up agreements with development partners.

Partner liaison group
: Day to day management of partner relationships and responsible for the import of code from development partners.

Specific roles that are active in the development process are:

Build co-ordinator
: Oversees mainline submissions. Engineers create pending changelists which are submitted once the build co-ordinator approves. The co-ordinator checks the outcome of the build and follows up any problems. There are about 15 of these people working in rota, drawn from the development community.

Release engineer
: Most project teams will have specific people responsible for managing submissions to team development codelines. Release projects also assign engineers to manage the release codeline.

Partner liaison engineer
: Responsible for accepting deliveries from development partners, following any legal process required and integrating accepted code into the repository.

This may seem like a lot of people, but bitter experience tells us that they are all necessary for successful partnership projects. It is very easy for standards to slip unless there are people who make it their business to maintain them. The standards only exist at all if they are agreed by all parties and this is best done near the beginning of a partnership.

A common scenario in our industry is for naive managers to assume that development partners solve resource problems without causing overheads. This is often called "outsourcing" and can be a disaster unless attention is paid to the framework of the relationship.

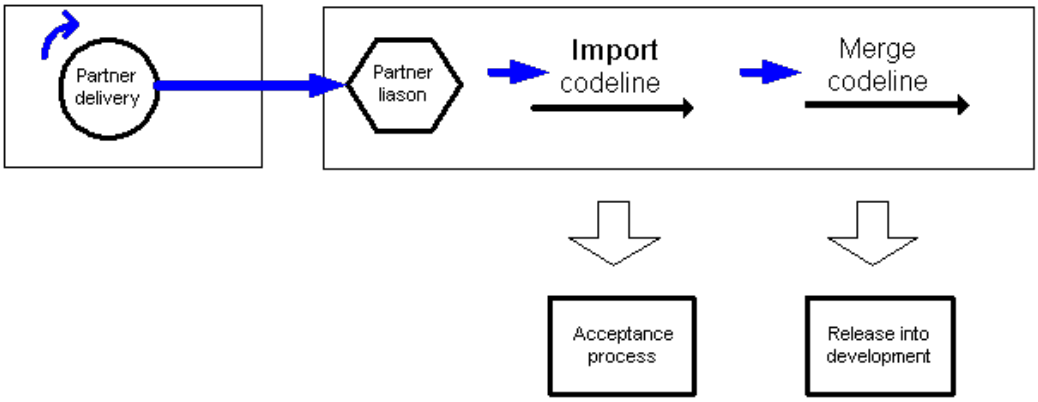
Quality and formal deliveries

There are broadly two categories of code coming into Symbian during the development process:

1. Code under joint development;
2. "Off the shelf" code.

Joint development

Code under joint development falls into the pattern described in the *codeline patterns* section above. Usually, there is an agreed delivery schedule in which the development partner provides an integrated delivery of code that will have been tested by them beforehand. On receipt of this code, the partner liaison engineer stores the code in an import codeline and carries out a formal acceptance test, communicating its results to the supplier. The code is then merged with Symbian code, with the correct IPR categorisation applied and made available to developers.



Assuming this isn't a final delivery, a separate process will deliver a new baseline to the development partner so that ongoing work is in an up to date context.

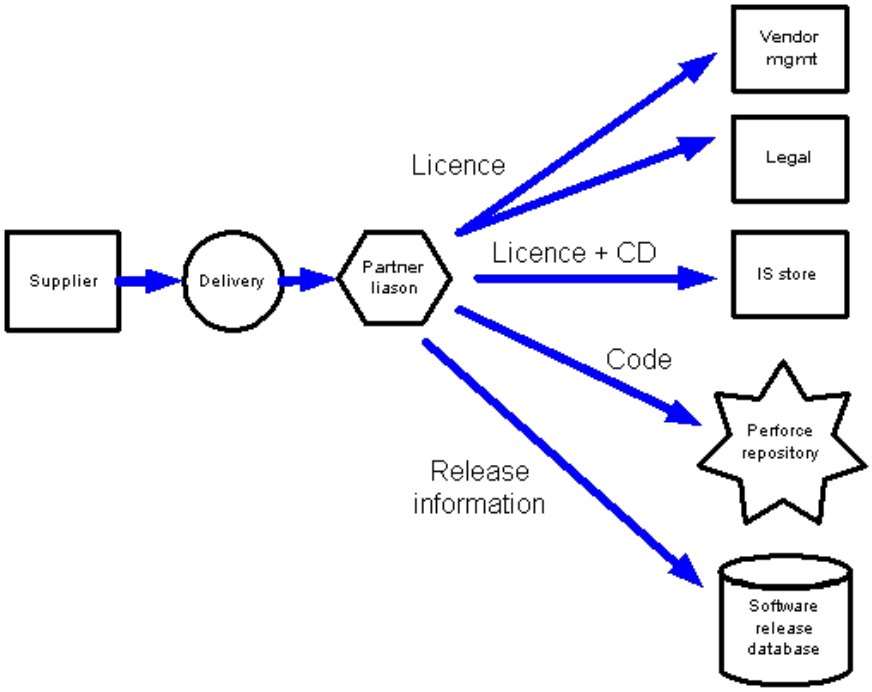
Note that this process works best if the partner's environment matches Symbian's own. We try to ensure that the build framework and tools can be exported and used by the partner without modification.

The process also assumes that:

a. There is a well understood delivery mechanism which includes clear identification of the software package;
b. There is a well defined test procedure used by both parties.

Off the shelf

"Off the shelf" deliveries have a simpler software engineering model but a more rigorous commercial process. This material has already been subject to evaluation, so there is no acceptance test involved.



The material delivered to developers through Perforce would normally be tagged as category B software. Much of the rest of the process is to ensure that we have proper records and archives of the delivery.

Licensee projects

So far, we've looked at development work, but Symbian is also itself engaged in working with mobile 'phone manufacturers on their own projects. Key characteristics of licensee projects are:

1. They are developed co-operatively by a potentially large number of companies that are normally geographically separated;
2. Each company must be free to choose which source control system it uses;
3. No single company will have access to all of the source to the project (due to IP restrictions).

Actually, this description applies somewhat to internal development as well, but the scale of a licensee project is greater. Also the component distribution network in development is like a star with Symbian as

its hub whereas for licensee projects it is more like a web.

From Symbian's perspective, the main differences between these projects and our internal development are:

- The product configuration is managed elsewhere;
- Symbian must protect licensee intellectual property internally.

In this context, Symbian is acting much like a component supplier, which in turn we hope gives us a good insight into the requirements such organisations have of the Symbian product.

As a result of these project characteristics, it is not possible to distribute development environments via a single source control system and a build process as would be classically done. We have therefore developed a binary distribution system. Arguably, even if we could use a single source control system and build process, we probably wouldn't want to, because environment distribution would be extremely slow (since each site would need to perform a build each time).

The key point of all this is that source control is not enough to manage these projects - you need an efficient and well-managed component distribution system too.

Intellectual property protection is achieved by managing the code for these projects in a different depot whose protection strategy is organised in terms of licensee groups. The main value of this is to ensure that one licensee's IP is not visible to another licensee's engineers who may be working onsite. It is also very difficult for engineers to accidentally integrate code from this depot to the one used for Symbian's own development.

Summary and open issues

Symbian derives great benefit from fostering co-operative development of its product. Successful operation of development partnerships involves establishing a good commercial and development framework from the beginning and providing the resources to operate it efficiently. Protection of intellectual property is a significant element of the framework.

Specific recommendations are:

1. Ensure that all projects have a configuration management plan and the resources to carry it out. In particular identify how third party code is to be managed and name the staff who have specific roles in this area.
2. Be serious about intellectual property - both yours and other people's. Actively manage it and keep records of what comes in and goes out.
3. Make specific agreements with development partners covering:
    - Delivery formats, schedules and labelling;
    - Intellectual property management;
    - Acceptance tests and processes.
4. Make sure that you and your development partners use the same build and diagnostic tools so that there are no surprises during acceptance which are due to environmental differences. This implies that build scripts are exportable - keep elements dealing with local infrastructure (such as Perforce commands) separate.
5. Be as rigorous about distribution management as you are about source management.
6. Make sure that your organisation has the infrastructure to actively support your relationship with development partners.

The costs attached to these recommendations will depend greatly on specific circumstances, but at Symbian we have about five people in the partner management team and a typical large project will use a major portion of one engineer's time on managing third party code.

Open issues for us are the desire to simplify the legalities wherever possible and to minimise delays and overheads involved in keeping internal and external engineers synchronised. This affects both the processes involved in the turnaround of changes between teams and the streamlining of binary distribution.

Glossary

2G, 2.5G, 3G
: In mobile telephony, second-generation protocols use digital encoding and include GSM, D-AMPS (TDMA) and CDMA. 2G networks are in current use around the world. These protocols support high bit rate voice and limited data communications. They offer auxiliary services such as data, fax and SMS. Most 2G protocols offer different levels of encryption. 2.5G protocols extend 2G systems to

provide additional features such as packet-switched connection (GPRS) and enhanced data rates (HSCSD, EDGE). 3G protocols support much higher data rates, measured in Mbits/second, intended for applications other than voice. 3G networks are expected to be starting in Japan by 2001, in Europe and part of Asia/Pacific by 2002, and in the US later. 3G will support bandwidth-hungry applications such as full-motion video, video-conferencing and full Internet access. For more information see http://www.symbian.com/technology/glossary.html on Symbian's web site.

Catchup integration

Part of a common codeline pattern where a development codeline is brought into step with the mainline prior to integrating completed changes to the mainline (the integration to mainline is called a Publish integration).

Component

A defined part of the system. Components usually represent a specific piece of functionality and are defined both in terms of a unit of source and a set of distributable binary objects.

IP

Intellectual Property.

IPR

Intellectual Property Rights.

Symbian OS

The brand name of the operating system that Symbian produces. Formerly known as EPOC.
