

# Experiences with certification of reusable components in the GSN project in Ericsson, Norway

**Parastoo Mohagheghi** (*Ph.D. Student, NTNU*)

Ericsson AS, Grimstad,  
Norway  
Tel + 47 37.293069, Fax +47 37.043098  
[etopam@eto.ericsson.se](mailto:etopam@eto.ericsson.se)

**Reidar Conradi**

Dept. Computer and Information Science  
NTNU, NO-7491 Trondheim, Norway  
Tel +47 73.593444, Fax +47 73.594466  
[conradi@idi.ntnu.no](mailto:conradi@idi.ntnu.no)

## ABSTRACT

Software reuse, or component-based development is regarded as one of the most potent software technologies in order to reduce lead times, increase functionality, and reduce costs. The Norwegian INCO R&D project (INcremental and COmponent-based development) aims at developing and evaluating better methods in this area [9]. It involves the University of Oslo and NTNU in Trondheim, with Ericsson as one of the cooperating industrial companies.

In this paper we discuss the experiences with the process to identify, develop and verify the reusable components at Ericsson in Grimstad, Norway. We present and assess the existing methods for internal reuse across two development projects.

## Keywords

Software reuse, Components, Layered system architecture, Software quality, Quality requirements.

## 1 INTRODUCTION

Companies in the telecommunication industry face tremendous commercial and technical challenges, characterised by very short time to market, high demands on new features, and pressure on development costs to obtain highest market penetration. For instance, Ericsson has world-wide adopted the following priorities: *faster, better, cheaper* – in that order. Software reuse, or component-based development, seems to be the most potent development strategy to meet these challenges [2][8]. However, reuse is no panacea either [4].

When software components are developed and reused *internally*, adequate quality control can be achieved, but the lead time will increase. Newer development models, such as incremental development, are promoting reuse of ready-made, *external* components in order to slash lead times. However, external COTS (Components-Off-The-Shelf) introduce new concerns of certification and risk assessment [1]. Both internal and external reuse involves intricate (re)negotiation and prioritisation of requirements, delicate compromises between top-down and bottom-up architectural design, and planning with not-yet-released components (e.g. middleware).

The present work is a pre-study of reuse in the GSN (GPRS Support Node, where GPRS stands for General Packet

Radio Service) project [6], and where Ericsson in Grimstad, Norway is one of the main participants. We present and assess the existing methods for component identification and certification at Ericsson in Grimstad for reuse across several projects.

In the following, section 2 presents the local setting. Section 3 introduces the reusable components while section 4, 5 and 6 discuss the quality schemes for reusable components and certification. Section 7 summarises experiences and aspects for further study.

## 2 THE LOCAL SETTING AT ERICSSON AS

Ericsson is one of the world's leading suppliers of third generation mobile systems. The aim of software development at Ericsson in Grimstad is to build robust, highly available and distributed systems for real-time applications, such as GPRS and UMTS networks. Both COTS and internal development are considered in the development process. The GSN project at Ericsson has successfully developed a set of components that are reused for applications serving UMTS networks. To support such reuse, the GSN project has defined a common software architecture based on layering of functionality and an overall reuse process for developing the software.

Figure 1 shows the *four GSN architectural layers*: the top-most application-specific layer, the two common layers of business-specific and middleware reusable components, and the bottom system layer. Each layer contains both internally developed components and COTS.

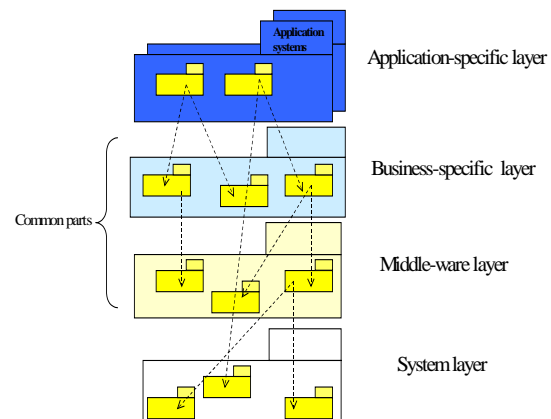


Figure 1. GSN application architecture with four layers.

Application systems use components in the common part. Applications address functional requirements, configuration of the total system and share components in the business-specific layer. The middleware layer addresses middleware functionality, non-functional requirements and what is called *system functionality* (to bring the system in an operational state and keep it stable). It also implements a framework for application development.

Application systems sharing this reusable architecture are nodes in the GPRS or UMTS network, both developed by Ericsson AS, the former in Norway and the latter in Sweden. However, the process of identifying reusable components up to the point that they are verified and integrated in a final product, still has shortcomings. Focus in this article is on certification of reusable components in the middleware and business specific layers in Figure 1, what we have called for “common parts” in short.

### 3 THE REUSABLE ARTIFACTS

The most important reusable artifact is the software architecture. By (software) architecture we mean a description/specification of the high-level system structure, its components, their relations, and the principles (strategies) and guidelines that govern the design and evolution of the system. The system architecture description is therefore an artifact, being the result of the system design activity.

Middleware is also an artifact that is reused across applications. It addresses requirements from several applications regarding non-functional requirements and traditional middleware functionality. Several business-specific components are also reusable.

Because of shared functional requirements, use cases and design artifacts (e.g. patterns) may be reused as well. The development process consists of an adaptation of RUP [7], a quality scheme, and configuration management (CM) routines. This process (model) is also a reusable artifact.

We can summarise the *reusable artifacts* as:

- A layered architecture, its generic components and general guidelines.
- Reusable components are either in the business-specific or middleware layers (both internally developed, and called common parts in Fig. 1), or in the basic system layer. Components in the business-specific or middleware layers are mostly written in the proprietary Erlang language, a real-time version of Lisp, and contain almost half part of the total amount of code written in Erlang. The system layer is a platform targeted towards wireless packet data networks containing hardware, operative systems and software for added features.

- Architectural (i.e. design) patterns and more specific guidelines.
- Partly shared requirements and use cases across applications.
- Common process, based on an adaptation of RUP and including a quality scheme and CM routines -see below.
- A development environment based on UML.
- Tools as test tools, debugging tools, simulators, quality assurance schemes.

The adaptation of the *RUP process* is a joint effort between the GPRS and UMTS organisations in Ericsson. It covers tailoring of subprocesses (for requirement specification, analysis and design, implementation, test, deployment and CM), guidelines for incremental planning, what artifacts should be exchanged and produced, and which tools that should be used and how.

To give a measure of the software complexity, we can mention that the GPRS project has almost 150 KLOC (1000 lines of code excluding comments) written in Erlang, 100 KLOC written in C and 4 KLOC written in Java. No figures are available for the number of reusable components but the applications share more than 60% of the code.

### 4 THE QUALITY SCHEME FOR THE ARCHITECTURE

The architecture was originally developed to answer the requirements for a specific application (GPRS). Having reuse in mind (between different teams in different organisations), the approach has later been to develop and evolve architectural patterns and guidelines that are reusable also to UMTS applications.

With requirements we mean both functional requirements and non-functional requirements. The latter are called quality requirements in [3], and are either development requirements (e.g. maintainability and reusability) or operational requirements (e.g. performance and fault-tolerance). While it is possible to map functional requirements to specific components, quality requirements depend on architecture, development process, software quality and so on. The architecture should meet all these requirements.

The process of identifying the building blocks of the architecture has partly been a *top-down* approach with focus on functionality, as well as performance, fault-tolerance, and scalability. A later recognition of shared requirements in the extended domain (here UMTS) has lead to a *bottom-up*, reverse engineering of the developed architecture to identify reusable parts across applications.

This implies a joint development effort across teams and organisations. However, we do not yet have a full-fledged product-line architecture.

Some important questions to verify reuse of the architecture are:

- How well can the architecture and components for a specific product meet the requirements for other products? The answer may lie in the degree of shared requirements. The project has succeeded to reuse the architecture, generic components and patterns in such a wide degree that it justifies investments considering development with reuse.
- How well are the components documented? How much information is available on interfaces and internal implementations? As mentioned initially, this is easier to co-ordinate when components are developed inside Ericsson and the source code is available. Nevertheless one of the most critical issues in reuse is the quality of the documentation which should be improved. The Rational UML tool is used in the development environment and all interfaces, data types and packages are documented in the model. In addition guidelines, APIs (Application Programming Interfaces) and other documentation are available.
- How well the developed architecture meets the operational requirements in the domain? This has been based on knowledge of the domain and the individual components, overall prototyping, traffic model estimations, intensive testing, and architectural improvements.
- How well the developed architecture meets the development requirements? It is not easy to answer as measuring the maintainability or flexibility of an architecture needs observations over a time. But we mean that the developed architecture has the potential to address these aspects. This is discussed more in the coming chapter.

As mentioned, design patterns and guidelines are also considered part of the architecture. A design pattern is a solution to a common problem. Hence when similarities between problems are recognised, a verified solution is a candidate for generalisation to a pattern. This solution must however have characteristics of a reusable solution regarding flexibility, design quality, performance etc. A large number of patterns are identified and documented for modelling, design, implementation, documentation or test. Based on the type of pattern, different teams of experts should approve the pattern.

## 5 CERTIFICATION OF THE ARCHITECTURE REGARDING QUALITY REQUIREMENTS

The architecture is designed to address both functional and quality (non-functional) requirements. While the functional requirements are defined as use cases, quality requirements are documented as the *Supplementary Specifications* for the system. One of the main challenges in the projects is the task of breaking down the quality requirements to requirements towards architecture, components in different layers or different execution environments. For instance a node should be available for more than 99.995% of the time. How can we break down this requirement to possible unavailability of the infrastructure, the platform, the middleware or the applications? This is an issue that needs more discussion and is not much answered by RUP either.

All components should be optimised be certified by performing inspections and unit testing. When the components are integrated, integration testing and finally target testing are done. The project however recognised that the architecture and the functionality encapsulated in the middleware layer (including the framework) address most of the quality requirements. The first step is then to capture the requirements towards architecture and the middleware layer:

- In some cases, a quality requirement may be converted to a use case. If such a conversion is possible, the use case may be tested and verified as functional use cases. For example the framework should be able to restart a single thread of execution in case it crashes.
- Other requirements are described in a Supplementary Specification for the middleware. This contains the result of breaking down the quality requirements towards the node when it was possible to do so, requirement on documentation, testability, etc.

Discussion on how to best capture quality requirements is still going on.

Quality requirements as performance and availability are certified by development of scenarios for traffic model and measuring the behaviour, simulation, and target testing. The results should be analysed for architectural improvements. Inspections, a database of trouble reports and check lists are used for other requirements as maintainability and documentation.

The architecture defines requirements to applications to adopt a design pattern or design rule to fulfil quality requirements as well.

The final question is how to predict the behaviour of the system for quality requirements? Domain expertise, prototyping, simulations and early target testing are used to answer this. Especially it is important to develop incrementally and test as soon as possible to do adjustments, also for the architecture.

## 6 THE QUALITY SCHEME FOR DEVELOPING NEW COMPONENTS

The process for software reuse is still not fully organised and formalised. When the decision for reuse is taken, the development process (RUP) should be modified to enhance the potential for reuse. The current process is summarised in the following steps:

- a) The first question when facing a new component is how generic this component will be. The component may be placed in the application-specific layer, the business-specific layer (reusable for applications in the same domain), or the middleware layer (the most generic part).
- b) If the component is recognised to be a reusable one:
  - Identify the degree of reusability.
  - Identify the cost of development to make the component reusable (compared to the alternative of developing a solution specified and optimised for a specific product).
  - Identify the cost of optimisation, specialisation and integration, if the component is developed to be more generic.
- c) Develop a plan for verifying the component. This depends on the kind of component and covers inspections, prototyping, unit testing and system testing, before making it available as a reusable part by running extra test cases. A complete verification plan may cover all these steps.

When the reuse is across products and organisations in Ericsson, a joint team of experts (called the Software Technical Board, SW TB) takes the decision regarding shared artifacts. The SW TB should address identification to verification of the reusable component and together with the involved organisations decide which organisation *owns* this artifact (should handle the development and maintenance). Teams in different product areas support the SW TB.

## 7 EXPERIENCES AND SUGGESTIONS FOR FURTHER IMPROVEMENTS

As mentioned, an adaptation of RUP has been chosen to be the development process. The development is incremental where the product owners and the software technical board jointly set priorities. Reuse is recognised to be one of the most important technologies to achieve reduced lead time, increased quality, and reduced cost of development. Another positive experience with reusable process, architecture and tools is that organisations have easier access to skilled persons and shorter training periods in

case of replacements.

Some aspects for further consideration regarding reuse are:

1. Improving the process for identifying the common components. This is mainly based on expertise of domain experts rather than defined characteristics for these components.
2. Coupling the development of common parts to the development plan of products using them.
3. Finding, adopting, improving or developing tools that makes the reuse process easier. An example is use of the multi-site Clearcase tool for configuration management of files.
4. Improving and formalising the RUP-based, incremental development process and teaching the organisation to use the process. This is not always easy when development teams in different products should take requirements from other products into consideration during planning. Conflicts between short-time interests and long-term benefits from developing reusable parts must be solved, see e.g. [5].
5. Developing techniques to search the developed products for reusable parts and improving the reuse repository.
6. Define a suitable **reuse metrics**, collect data according to this, and use the data to improve the overall reuse process.

The topic of certifying the architecture and the system regarding quality requirements should be more investigated and formalised. Some aspects are:

1. Improve the process of breaking down the quality requirements.
2. Improve the development process (an adaptation of RUP) on how to capture these requirements in the model or specifications.
3. Improve planning for certification of quality requirements. While functional requirements are tested early, test of quality requirements has a tendency to be delayed to later phases of development, when it is costly to change the architecture.

## 8 CONCLUSION

Implementing software reuse combined with incremental development is considered to be the technology that allows Ericsson to develop faster, better and cheaper products. However, future improvement of the technology, process, and tools is necessary to achieve even better results. The INCO project aims to help Ericsson in measuring, analysing, understanding, and improving their reuse process, and thereby the software products.

## 9 REFERENCES

1. Barry W. Boehm and Chris Abts: "COTS Integration: Plug and Pray?", IEEE Computer, January 1999, p. 135-138.
2. Barry W. Boehm et al.: "Software Cost Estimation with Cocomo II (with CD-ROM)", August 2000, ISBN 0-130-26692-2, Prentice Hall, 502 p. See also slides from the FEAST2000 workshop in 10-12 July, 2000, Imperial College, London, [http://www-dse.doc.ic.ac.uk/~mml/f2000/pdf/Boehm\\_keynote.pdf](http://www-dse.doc.ic.ac.uk/~mml/f2000/pdf/Boehm_keynote.pdf).
3. Jan Bosch: "Design & Use of Software Architectures: Adopting and evolving a product line approach", Addison-Wesley, May 2000, ISBN 0-201-67494-7, 400 p.
4. Dave Card and Ed Comer: "Why Do So Many Reuse Programs Fail", IEEE Software, Sept. 1994, p. 114-115.
5. John Favaro: "A Comparison of Approaches to Reuse Investment Analysis", Proc. Fourth International Conference on Software Reuse, 1996, IEEE Computer Society Press, p. 136-145.
6. GPRS project at Ericsson: <http://www.ericsson.com/3g/how/gprs.html>
7. Ivar Jacobson, Grady Booch, and James Rumbaugh: "The Unified Software Development Process", Addison-Wesley Object Technology Series, 1999, 512 p., ISBN 0-201-57169-2 (on the Rational Unified Process, RUP).
8. Guttorm Sindre, Reidar Conradi, and Even-André Karlsson: "The REBOOT Approach to Software Reuse", Journal of Systems and Software (Special Issue on Software Reuse), Vol. 30, No. 3, (Sept. 1995), p. 201-212, <http://www.idi.ntnu.no/grupper/su/publ/pdf/jss.df>.
9. Dag Sjøberg / Reidar Conradi: "INCO proposal for NFR's IKT-2010 program", 15 June 2000, Oslo/Trondheim, 52 p., <http://www.idi.ntnu.no/grupper/su/inco.html>.

