A Tool for Cynical Reengineering

Kristoffer Kvam, Rodin Lie and Daniel Bakkelund

Telenor, CRM, Business Logic,
kristoffer.kvam@telenor.com
rodin.lie@telenor.com
daniel.bakkelund@telenor.com

Abstract. In this paper the XRadar[2] system analysis report framework is presented. XRadar is an extensive open source framework for analysing and monitoring the evolution of software systems. The solution is proven on a large middleware platform giving increased control and being an important tool in a reengineering process. The requirements for developing the framework is presented and various extensions and areas of use are discussed. Finally the future plans for the framework is introduced.

Keywords: Reengineering, Refactoring, Software Metrics, Open Source, Analysis Tool, XSL, XML

1 Introduction

1.1 Challenge

When the developers and architects are claiming "We have to refactor parts of this system", management often responds: "That might be a good idea, why don't you make a business case. You need to document why that increases the system's business value, what to start with, how to report progress and find a way to avoid fallback".

Facing a challenge similar to this, we started looking for tools that could kick-start us. We tested different commercial products, open source tools and in-house developed analysis tools. They all gave us important information in terms of complexity, redundant code, code duplication, dependencies, cycles, test coverage, performance and so on. This bottom up view was helping but we needed one common top down view of the system - preferably with drilldown functionality within the various areas.

We also realized that another essential issue was not properly addressed in the tools we found. Imagine a stock market analyst making a recommendation on buying a stock. The analyst would seem somewhat ignorant if he only looked at the fundamentals supporting the stock. A lot of his attention would most likely on how the stock has traded back in time : its history. In reengineering we face the same challenge. A metric often gives little meaning before you look at how it has been developing.

Hence, we were led to the development of the XRadar - an analysis tool and plug-in framework for making cynical reengineering [1] decisions. This paper covers the evolution to its current release, its added value, its architecture and its future.

1.2 Telenor

Telenor is Norway's largest telecommunications company with numerous international interests.

COS is a middleware system designed to give front-end applications a consistent view across multiple back-end systems. There are over 20 front end applications serving retail outlets, customer support, large corporate customers and internal functions. The back end systems include Sybase and Oracle databases, network connections and mainframes, all of which are logically interconnected through the use of batch jobs, scripts and database stored procedures.

COS has evolved over 5 years into a large system, composed of many subsystems. After a period of sustained development the problems were manifold. The developers and architects on the system was more and more often expressing the need for reengineering.

The Pareto project was instigated with solving these problems. The XRadar framework was developed as a sub-project.

2 Requirements

Subjective arguments seldom convince a manager to invest in reengineering or refactoring. By providing hard facts about the system the problems are easier revealed and documented by objective reports. Developers, architects and project management all need information, preferably in a structured form. This motivates the requirements to the system.

Customized View for all Stakeholders Software systems of some size need to show its different faces. All stakeholders (developers, architects and managers) need specialized reports within their areas. Combined and weighted metrics should give them advice on where to focus. Maintaining spreadsheets is time-consuming and boring. Spending some time configuring reports that can be automated and repeated at fixed intervals leaves more time for quality work.

Historical View of the Systems Development When educating architects and developers it's important to be able to look at a system's history. Negative trends can be discovered early and corrected by proper education of the developers or by doing architectural changes. Furthermore, when reengineering, monitoring the progress over time is essential to make the right decisions.

Drilldown When analysing a large system, you need to be able to see the system from various levels of detail. You have the obvious drilldown possibilities representing the logical properties of the system: total system, packages, classes and methods. But often these do not directly represent how one views the system as a set of entities. More often you speak of system areas, modules or subsystems, represented by a group of packages. In our drilldowns, we wanted to be able to view the system from this subsystem perspective as well.

Layering Having split the system into subsystems, a second goal is to analyse the layering of the system. You want to define how the subsystems in their layers legally may interact and then analyse how the system actually fits this ideal picture. Some of these illegal dependencies causes cycles - a very bad symptom in an architecture. When a dependency error is detected, you also want to pinpoint what causes it to do a proper fix.

Flexible Framework Due to changing system focus, it would be highly beneficiary to be flexible on the needs of the future. Furthermore, one systems key indicator may not be available for another system. Hence, flexibility is a key issue for our tool:

- A framework for combining metrics into different status reports
- A framework for easily plugging in more metric sources as the need arises
- A framework for easily adding new reports
- A framework that is independent on the system being analysed, while at the same time being open to system specific changes

3 XRadar

The XRadar is our solution to the requirements. Presently it only supports Java, but there are plans to produce plug ins for other leading languages. Although developed as a corporate tool, it has been decided to make it open source. It heavily relies on open source products, and in this way we can give something back. In the following subsection the XRadar is explored from a funtional as well as an architecural view point. Thereafter follows some details on the ease of extending the XRadar to a system's specific purposes. Finally, we will present the future plans for the XRadar.

3.1 Measurements

As default, the XRadar gives measurements on standard software metrics such as :

- package metrics and dependencies
- code size and complexity
- coding violations

– code-style violations

Data from unit test metrics and code coverage are also integrated, but must be obtained running the test suites on the system while doing monitoring. Internally we have also integrated even more measurements such as from source control, performance metrics and SQL procedures. It is not unlikely that similar plug ins will be made available for the public in the future.

3.2 Reports

The XRadar is available in two forms. XRadar Statics gives reports on the current build of the system. The other form, XRadar Dynamics, includes the time dimension and views the historical and present versions along the time axis. The Dynamics version relies on a set of two or more Statics runs of different system releases to work properly.

XRadar Statics The HTML presentation starts at the system-subsystem level, but drilldowns are available down to package, class and method level. Navigation can be done in "javadoc style" to find suitable packages and classes. In the main window you can view scorecard reports on architecture, package design, code, test and system specific views. Reports consist of tables with metrics and hyperlinks for detail, and dynamic SVGs with possible navigation features. To give an example, one important view in the scorecard section, the architecture dependency report is defined by the dependency configuration. The graphical part of that report is shown in Fig. 1.

There is also a system analysis section in the XRadar. Here you are able to view reports on system design, code quality, class coupling, redunant code, external dependencies as well as the system specific reports. Several of these can be used to find trouble areas across the system.

XRadar Dynamics The Dynamics report is a little more coherent in its analysis. It gives one view for all levels of detail. Drill-downs are possible from the system level to subsystem level and package level. All views include selected graphical trend reports on architecture and package design, code size and quality and test quality. Along the time axis you have the releases. For more historical data within one area, there is also more data in tabular form. See Fig. 2 for some example graphs.

3.3 Architecture

The XRadar is a term describing the HTML/SVG reports that we call Statics and Dynamics. The framework we use to produce these reports is called the XRadar analysis framework. There is nothing magical to this framework - it is based on best practices within xml processing. When you run Statics you initiate an XSL pipeline where each metric source is integrated by a specific merge style

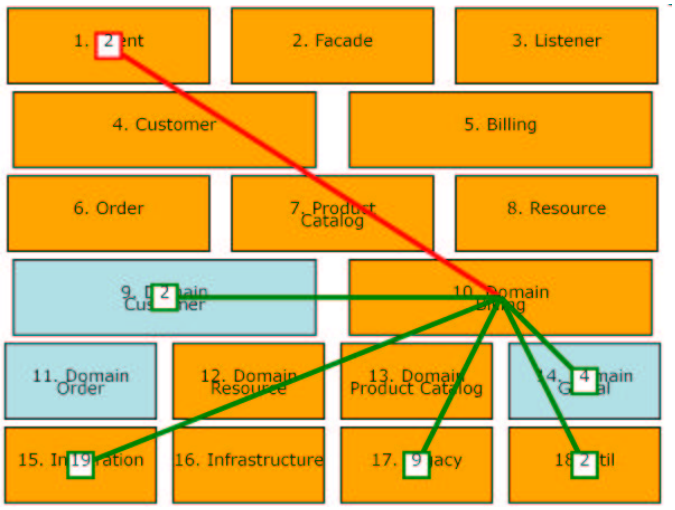

Fig. 1. An example svg images of the dependency graph in the XRadar Statics html report. Here the cursor is positioned over one of the subsystems and it shows the legal and illegal dependencies from that particular subsystem.

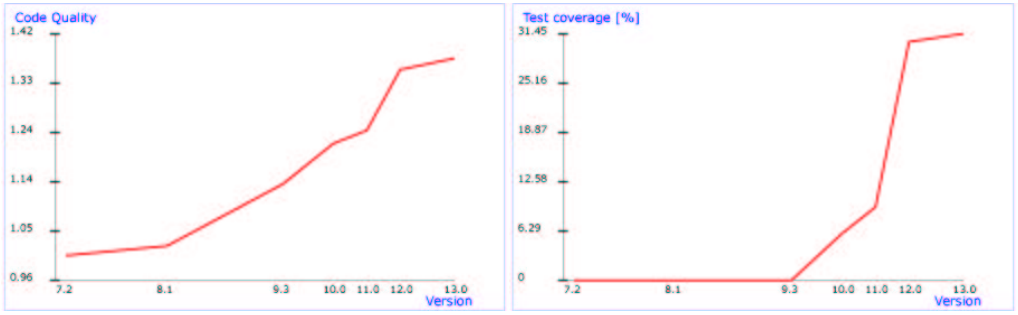

Fig. 2. Example SVG views of the development of code quality and test coverage in the XRadar Dynamics html report.

sheet. Relevant data from each merge is added to the total xml report. In a way you produce an extendable xml metric database of the system. The final XML report is post processed and then sent to the relevant report framework. For the XRadar Statics report, that report framework is to a set of XSL style sheets that produces the final HTML/SVG report. See Fig. 4 for a rough figure of the framework running XRadar Statics.

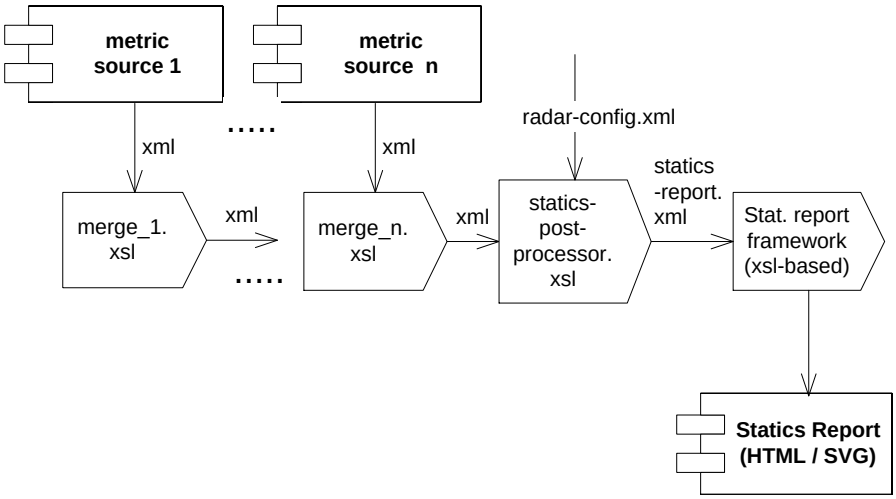

Fig. 3. The XRadar Statics pipeline

The XRadar architecture for Dynamics is very similar as for Statics. A pipeline is initiated, but here each source to be merged is a final XRadar Statics XML report. Each is merged sequentially through a pipeline producing a common historical XML database of the system. This XML is then sent to the XRadar Dynamics report framework that produces the final HTML/SVG report. See Fig. 4 for graphical representation of the pipeline.

The XRadar framework currently runs through a build file in Apache Ant [3] as a batch process, but there is no reason why it cannot be initiated and controlled by any other means that offer the needed supporting functionality. See the *Future Work* section below from more on this.

3.4 Configuration

You have one property file were you set path configurations. Additionally you are also able to configure the system into subsystems in a dedicated XML file. You do this to make it run to its full potential. This is done by defining specific

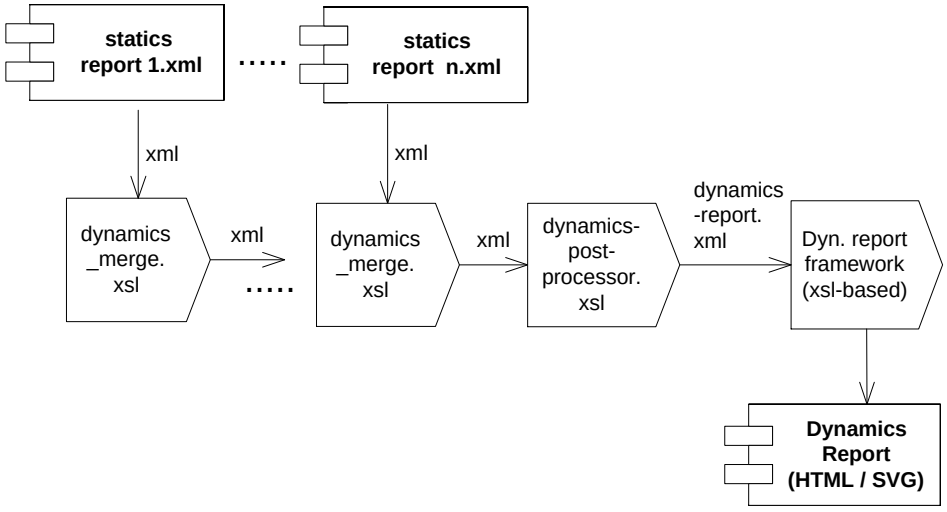

Fig. 4. The XRadar Dynamics pipeline

packages belonging to a subsystem. When doing that you must also decide how the subsystems are allowed to depend on each other. This configuration is added to the XRadar in the post processing part of the Statics pipeline. This integration can be seen in Fig. 3 were you can see that the radar-config.xml data is merged in the postprocessor.

3.5 How to make a plug in

Making a plug in for a new data source consists of a couple of steps. If we assume that you have an analysis system that you want integrated in the XRadar framework, the only prerequisite is that the system produces XML and that the library in some way can be initiated from ant. The following steps must be taken:

1. To run the analysis you need to add one more target to the system specific analysis task. You can have both a library that analyses the source or simply a process that gets XML from a loosely coupled external source.
2. To merge the analysis you need to add one more target to the system specific merging task. This should call your own developed XSL merging template. The report is now merged with the common Statics XML analysis database.
3. To view the data in Statics you need to edit one of the system specific XSL report templates in the XRadar. These are explicitly separated out in the XRadar. They are as a default empty, and can be edited. Here you are free to so whatever you need for your report.
4. To view the data in Dynamics, you only need to edit one of the available system specific templates for the Dynamics report. The reason why you most

probably do not need any merge is that if the data is included in the XML database in the proper way, they are automatically included in the XRadar.

3.6 Existing plug ins

The XRadar is currently bundled as one package with a set of open source sources for the measurements. The common denominator of these is that they all produce XML in some form so that they easily may be integrated into the XRadar. These produces the measurements as mentioned in the subsection above:

- JDepend [5] : Measures package dependecies and calculates relevant metrics.
- JavaNCSS [6] : Finds standard source code metrics such as source statements, method cyclomatic complexity and javadoc length.
- PMD [7] : Finds code violations in the source. This is a rule based system where you can define your own rules in addition to the ones that come with the library.
- Checkstyle [8] : Finds code style errors, but also includes some features similar to PMD

Of the optional plug ins that depend on external analysis you have :

- JUnit [9] : Standard unit test framework for Java.
- JCoverage [10] : Unit test code coverage tool.

4 Future work

There are several directions we can take the XRadar and the XRadar framework now that it has been made open source. Below, some ideas are mentioned.

Other languages One obvious direction is to integrate and build analysis and merge plug-ins so that other computer languages are supported. Hot candidates are C++ and C#. Most probably, several analysis tools for these languages exists and the job is to integrate these.

Other build frameworks There are concrete plans to support other build frameworks to run the XRadar pipeline in. Apache Maven [4] is a reasonable candidate in this respect. Through Maven you will be able to run XRadar out of the box on your already established projects using existing Maven plug ins for doing the analysis.

Other measurements and reports Internally we already have integrated reports on duplicated code, source control and production metrics. Several of these will be made public when proven.

More GUI models Since you have one common XML analysis database it is in principle no problem to add new analysis views. Today, the reports are two HTML reports: XRadar Statics and Dynamics. In the future you will likely also have a richer GUI client to be able run reports and analyse the system in real-time.

5 Conclusion

We have presented the XRadar framework in this paper. The tool is open source and open for extensions and refactorings. Loads of challenging opportunities are offered for those who wish to lift the framework and XRadar reports to new heights.

For us, the framework made a contribution to our success story, the Pareto Project. The tool will also prevent us to fall back to old sins. We hope the framework will give you the same advantage.

6 References

References

1. Kvam, K., Bakkelund, D., Lie, R Cynical Reengineering (2004) [To be presented at the conference XP 2004]
2. XRadar http://xradar.sourceforge.net/
3. Apache Ant http://ant.apache.org/
4. Apache Maven http://maven.apache.org/
5. JDepend http://www.clarkware.com/software/JDepend.html
6. JavaNCSS http://www.kclee.com/clemens/java/javancss/
7. PMD http://pmd.sourceforge.net/
8. Checkstyle http://checkstyle.sourceforge.net/
9. JUnit http://www.junit.org/index.htm
10. JCoverage http://www.jcoverage.com/