

A Case Study of Defect-Density and Change-Density and their Progress over Time

Anita Gupta, Odd Petter N. Slyngstad,
Reidar Conradi, Parastoo Mohagheghi
*Department of Computer and Information
Science (IDI)
Norwegian University of Science and
Technology (NTNU)
{anitaash, oslyngst, conradi, parastoo} at
idi.ntnu.no*

Harald Rønneberg, Einar Landre
*Statoil KTJ/IT
Forus, Stavanger
{haro, einla} at statoil.com*

Abstract

We have performed an empirical case study, investigating defect-density and change-density of a reusable framework compared with one application reusing it over time at a large Oil and Gas company in Norway, Statoil ASA. The framework, called JEF, consists of seven components grouped together, and the application, called DCF, reuses the framework, without modifications to the framework. We analyzed all trouble reports and change requests from three releases of both. Change requests in our study covered any changes (not correcting defects) in the requirements, while trouble reports covered any reported defects. Additionally, we have investigated the relation between defect-density and change-density both for the reusable JEF framework and the application. The results revealed that the defect-density of the reusable framework was lower than the application. The JEF framework had higher change-density in the first release, but lower change-density than the DCF application over the successive releases. For the DCF application, on the other hand, a slow increase in change-density appeared. On the relation between change-density and defect-density for the JEF framework, we found a decreasing defect-density and change-density. The DCF application here showed a decreasing defect-density, with an increasing change-density. The results show that the quality of the reusable framework improves and it becomes more stable over several releases, which is important for reliability of the framework and assigning resources.

1. Introduction

There exist few published, longitudinal empirical studies on industrial systems. Many organizations obtain large amounts of data related to their process and products. However, the data analyses are not done properly, or the results are kept inside the organization [11]. Currently, we are studying the reuse process in the IT-department of a large Norwegian Oil & Gas company named Statoil ASA¹. We have collected data from trouble reports and change requests for three releases of seven reusable components in a framework called the Java Enterprise Framework (JEF), as well as three releases of one application called DCF which uses this framework “as-is”, without modification.

The purpose is assessing software quality and evolution of the reusable framework, as well as the application, and comparing these. Our quality foci have been defect-density defined as the number of defects divided by kilo lines of source code, and stability or change-density defined as the number of change requests divided by kilo lines of source code. We have chosen these two attributes for measuring software quality as they are part of the stated quality focus for the reuse program in Statoil ASA. Lower defect-density means that fewer corrections are needed. When it comes to change-density, stability is important to achieve stable evolution, hence allowing for stable resources being assigned to adapt and perfect the reusable framework. It is also important for the applications using the framework to experience fewer defects and changes in the framework. For the application, lower change-density may indicate

¹ ASA stands for “allmennaksjeselskap”, meaning Incorporated.

maturity of the product. In this way, these quality attributes can be used to partially model evolution, as they show how the quality of the reusable framework and the application evolves over time.

We have defined several research questions through this empirical study. The research interests were obtained from literature, and include how the defect-density and change-density for the reusable framework vs. the application evolve over several releases, as well as what the relation is between change-density and defect-density for the reusable framework, in comparison with the application over several releases. Our results show that the stability and quality of the reusable JEF framework increases over time, while the DCF application shows an improving quality, but lowered stability.

Future studies will be used to refine and further investigate the research questions presented here. This paper is structured as follows: Section 2 contains terminology, Section 3 presents related work, Section 4 introduces the context in Statoil ASA, and Section 5 discusses research background and motivation. Section 6 contains the results of our analysis of trouble reports and change requests, and Section 7 summarizes and discusses these results, while Section 8 concludes.

2. Terminology

The area of software maintenance and evolution covers software changes due to defects and enhancements, which are the study objects in this investigation. IEEE [5] defines software maintenance as “(...) *the process of modifying a component after delivery to correct faults, to improve performance or other attributes, or to adapt to a changed environment*”. Still, there is the opposing view that software maintenance starts prior to delivery of the software. When it comes to software evolution, different opinions exist and there is no overall agreement on a definition. Some see evolution as being encompassed by maintenance as a broader term [11]. Others see evolution as a separate stage in the lifecycle of the software [2]. Finally, there is the view that software evolution is the way software systems dynamically behave over their lifetimes as they are maintained and enhanced [1], meaning that evolution really can be seen as a much broader, encompassing term than software maintenance. Software reuse is a management strategy, in terms of development *for* and *with* reuse, where development *for* reuse refers to the generalization of components towards reuse, and development *with* reuse has to do with the inclusion of

these reusable components in new and future development [10].

At different points in the development cycle, a system goes through various types of testing (reliability, efficiency etc.) in distinct stages (such as unit, system, integration etc.). When abnormalities in the operation of a system are found, these are reported as *failures*. These failures are reported to developers through *failure reports*. A *fault* is an underlying problem in a software system that leads to a failure. *Error* is used to denote the execution of a passive fault which leads to incorrect behaviour (with respect to the requirements) or system state [4], and also for any fault or failure resulting from human activity [3]. Sometimes, *defect* is used in place of fault, error or failure, without distinguishing the origin or whether it's active or passive. In Statoil ASA, defects are called *incidents*, but we will nevertheless use the term defect, which is widely used in literature. Other changes to the software, such as adapting to new environments or platforms, implementing new or changed requirements, optimizing the system, as well as improving performance or maintainability, can be thought of as enhancements (as opposed to fixing defects to maintain status quo). These changes are reported as *scope changes* at Statoil ASA and called change requests by us.

In Statoil ASA [12], defect-density and change-density are used to measure the quality of software. Lower defect-density indicates improving quality over time. Change-density indicates the frequency of changes incurred on software components, and for this measure, a relative stability is important in terms of assigned resources to the development process. Statoil ASA wishes more stability for the reusable framework, but not necessarily for the application, as we would expect requirements to be changed or added more often throughout the lifetime of the application. We will be looking at defect-density and change-density, to discover the characteristics of the reusable framework and the application's maintenance and evolution over several releases. Our results will therefore be a contribution in that context.

3. Related work

Understanding the issues related to software maintenance and evolution, involving change made to software, has been a focus since the 1970s. The focus has been on finding the origins, frequency and costs in terms of effort for changes. These software changes or alterations are of importance, since they are responsible for a large part of the total software costs.

Nevertheless, they're unavoidable; the ability to change software in quick and reliable ways is necessary so that new business ideas can be implemented, and companies thereby stay competitive [2].

Ostrand and Weyuker [15] analyzed faults in 13 releases from an inventory tracking system at AT&T. They showed a small decrease in fault-density with decreasing component size. Also, components that have a high number of faults in one release remain fault-prone in following releases. Finally, they saw that newer components had higher fault-density than older components.

Fenton et al. [16] investigated the connection between the number of faults and module size in a release-based development setting, but were unable to relate fault-density and module size to each other. However, they did find that size weakly correlates with the number of faults in pre-release, but not post-release, editions.

Another study by Malaiya and Denton analyzed several studies, introducing a theory of an ideal module size which adheres to the scale of economy [17]. Their theory is based on two dimensions; (1) the division of software into modules and (2) their individual implementation. Their results show that in the former dimension, the number of faults decrease as module size increases (due to communication overhead, and interface faults are reduced). However, in the latter dimension the number of faults increases with the module size. The authors have combined these two dimensions, and concluded that there is an "optimal" module size. For modules that are larger than the optimal module size, fault-density increases with module size. For modules that are smaller than the optimal module size, fault-density decreases with the module size.

Basili, Briand and Melo investigated the impact of reuse in software quality and productivity in object-oriented systems [18]. They used a study population of graduate level students, who developed information systems with the waterfall model, and four varying degrees of reuse. Their results show that defect-density decreases linearly with the rate of reuse. The authors hence claim that this is strong evidence that reuse helps improve the quality of software.

The aforementioned studies include defects and changes, but have not investigated defect-density and stability over several releases in a similar fashion to our investigation. Following below are some studies more directly related to our research here.

Mohagheghi et al. [7] undertook a study to determine the impact of reuse on defect-density and stability in the context of reuse. In their study, reused

components had lower defect-density than the non-reused ones. They also found that the reused components had a higher number of defects of the highest severity before delivery than expected (according to the total distribution), but fewer defects post-delivery. Reused components were less modified (more stable) than non-reused ones between successive releases in the number of modified source lines of code. The study lacked detailed data for change-density as the number of change requests per LOC.

Another interesting empirical study done by Selby [8] evaluated software reuse by mining software repositories from a NASA software development environment. The study examined 25 software repositories ranging from 3000-112000 source lines of code. The results revealed that software modules reused without revision had the fewest faults, fewest faults per source line, and lowest fault correction effort. Software modules reused with major revisions had the highest fault correction effort and highest fault isolation effort, as well as the most changes, most changes per source line, and highest change correction effort.

4. The Statoil ASA setting

Statoil ASA is a large, Norwegian company, in the Oil & Gas industry. It is represented in 25 countries, has a total of about 25,000 employees, and is headquartered in Europe.

The central IT-department of the company is responsible for developing and delivering software, which is meant to give key business areas better flexibility in their operation. It is also responsible for the operation and support of IT-systems. This department consists of approximately 100 developers, located mainly in Norway. Since 2003, a central IT strategy of the O&S (Oil Sales, Trading and Supply) business area has been to explore the potential benefits of reusing software systematically. Statoil ASA has developed a customized framework of reusable components. This framework is based on J2EE (Java 2 Enterprise Edition), and is a Java technical framework for developing Enterprise Applications [14]. Statoil ASA has chosen to call this customized framework the "JEF framework".

This IT strategy was started as a response to the changing business and market trends, in order to provide a consistent and resilient technical platform for development and integration [12]. The strategy is now being propagated to other divisions within Statoil ASA. The actual JEF framework consists of seven separate components. These are: JEF Client, JEF Workbench, JEF Util, JEF Dataaccess, JEF SessionManagement,

JEF Security and JEF Integration; which can be applied separately or together when developing applications. There are three releases of the JEF framework, namely: Release 1, 2 and 3. All JEF releases prior to release 1 have been for internal development only, while release 1 is the first to be used in other development projects in Statoil ASA. Table 1 shows the size and release date of the three JEF releases.

Table 1: The size and release date of the three JEF releases

Release 1:	Release 2:	Release 3:
14. June 2005	9. September 2005	8. November 2005
16875 SLOC	18599 SLOC	20348 SLOC

This JEF framework is currently being reused in two applications at Statoil ASA. However, we have in this study only investigated one of these applications, namely DCF (Digital Cargo Files), due to available data set. The other application will be investigated in future studies. DCF is mainly a document storage application. It imposes a certain structure to the documents stored in the application, and relies on the assumption that the core part of the documents is based on cargo (load) and deal (contract agreement) data as well as auxiliary documents pertaining to these information entities. DCF is meant to replace the current practice of cargo files, which are physical folders containing printouts of documents pertaining to a particular cargo or deal. A “cargo file” is a container for working documents related to a deal or cargo, within operational processes, used by all parties in the O&S strategy plan at Statoil ASA. There are also three releases of the DCF application, namely: Release 1, 2 and 3. As we do not currently have data for subsystems of the DCF application, we will be comparing it with the JEF framework on an overall level. Subsystems of DCF will be investigated in future studies. Table 2 gives an overview of the size and release date of the three DCF releases.

Table 2: The size and release date of the three DCF releases

Release 1:	Release 2:	Release 3:
1. August 2005	14. November 2005	8. May 2006
20702 SLOC	21459 SLOC	25079 SLOC

4.1 Change requests data in Statoil ASA

When the need for a change is identified, a change request is written and registered in the Rational ClearQuest tool. Examples of change requests are:

- add, modify or delete functionalities (perfective changes)
- solve an anticipated problem (preventive changes)
- adapt to changes from other JEF component interfaces (adaptive changes)

A change request usually impacts only one of the seven JEF components, but may impact more than one. If a change request impacts several components, it is related to the category *General* to indicate that this change request impacts the JEF framework as a whole. All registered change requests can be exported as Microsoft Excel files. Each change request contains an ID, headline description, priority (indicates the priority level of the change request) given by both customer and developer (Critical, High, Medium or Low), estimated time to fix, remaining time to fix, subsystem location (one of the seven JEF components), system location (e.g. JEF, DCF etc.), as well as an updated action and timestamp record for each new state the change request enters in the workflow. Statoil ASA does not register release numbers for changes, but a change request is always marked with the date at inception. This date is consistent with the release that was currently under development at the time.

4.2. Trouble reports data in Statoil ASA

The handling of defects is very similar to that of change requests. When a defect is detected during integration/system testing or other testing activities, a trouble report is written and stored in the Rational ClearQuest tool. Defects here refer to fixing problems.

A defect also usually impacts only one of the seven JEF components, but may impact more than one. If a defect impacts several components, it is similarly related to the category *General* (see section 4.1). All registered trouble reports can also be exported as Microsoft Excel files, and each trouble report contains an ID, headline description, priority (indicating the priority level of the problem) given by the developer (Critical, High, Medium or Low), severity that indicates how critical the problem is evaluated by developers (Critical, High, Medium or Low), classification (Error, Error in other system, Duplicate, Rejected and Postponed), estimated time to fix, remaining time to fix, subsystem location (one of the

seven JEF components), system location (e.g. JEF, DCF etc.), as well as an updated action and timestamp record for each new state the defect enters in the workflow. Trouble reports also do not include data on releases, but here too the date marked at inception does correspond with the release that was currently under development at that time.

An advantage of the close similarities between trouble reports and change requests is that Statoil personnel can easily switch from working with one to the other without requiring extra training. This saves time and resources when reshuffling of responsibilities becomes necessary.

5. Research method and Research Questions

Our motivation here is to investigate the issues surrounding defect-density and change-density, in relation to software quality, maintenance and evolution of the reusable JEF framework and the DCF application, and how these are handled in Statoil ASA. Specifically, we want to explore the way software defect-density and change-density evolves, for the reusable JEF framework and the DCF application, over several releases. We also want to explore the possible relationship between these two measures. In the following section, our research method and questions are presented.

The process of choosing research questions for our study has been done by using a combined top-down and bottom-up process. RQ1 and RQ2 was found from the literature (top-down), while RQ3 was chosen after the pre-analysis we did on the data collected from ClearQuest (bottom-up). As mentioned earlier, we have chosen to focus on defect-density and change-density for measuring software quality, as they are part of the stated quality focus for the reuse program in Statoil ASA. The purpose of this study is to gain initial understanding of the software maintenance and evolution in Statoil ASA from the viewpoint of these quality attributes. The following is a presentation of the research questions.

RQ1: How does the defect-density for the reusable framework vs. the application evolve over several releases? A study of RQ1 is important since it gives us the possibility to see if a higher quality level is achieved for the reusable framework. Former literature [7] [8] indicates that reused components have a lower defect-density than non-reused ones. The purpose here is to confirm whether the results from [7] [8] can be seen for the reusable JEF framework vs. the DCF application in Statoil ASA.

RQ2: How does the change-density for the reusable framework vs. the application evolve over several releases? A study of RQ2 is important, since we get the chance to see if stable evolution and hence stable resources are being assigned to adapt and perfect the reusable framework. Previous research [7] showed that reusable components have a lower code modification rate (are more stable) than non-reused components. The study also defined a hypothesis regarding change-density that was evaluated in this context. Our purpose here is to see whether the reusable JEF framework in Statoil ASA has a higher or lower change-density than the DCF application over several releases.

RQ3: What is the relation between change-density and defect-density for the reusable framework, in comparison with the application? Selby [8] in his study showed that modules reused without revision had the fewer changes in terms of either total changes or total changes per source line, as well as less change correction effort in terms of total change correction hours. While Selby defined changes collectively to include both faults and enhancements, we are in our study looking at these separately, in terms of change requests and trouble reports. Here, we are here interested in exploring the relation between change-density and the defect-density for the JEF framework and the DCF application in Statoil ASA.

6. Results of the analysis

In this section, we summarize the results from our analysis of defect-density and change-density for the JEF framework and the DCF application. Microsoft Excel was used to count defects and changes for the three JEF releases, and for the three releases of the DCF application. We have divided each JEF and DCF release, and their respective defects and changes, according to their respective release date. In total, there are 223 trouble reports and 205 change requests for the three JEF releases. The majority of change requests and trouble reports for JEF releases come from the first release. In total, there are 1140 trouble reports and 127 change requests for the DCF application. Again, majority of change requests and trouble reports for the DCF application come from the first release.

RQ1: How does the defect-density for the reusable framework vs. the application evolve over several releases? For the DCF application, 174 out of 1140 trouble reports have been related to other applications than DCF itself. Therefore, these 174 trouble reports are excluded from our analysis. In total, we have used 223 trouble reports for the JEF

framework and 966 trouble reports for the DCF application in the analysis of RQ1. We plotted our data in a lineplot, as shown in Figure 1. From this figure we can see that the defect-density for the JEF framework

decreases over the three JEF releases, and for the DCF application this is also the case. However, the reusable JEF framework has significantly lower defect-density than the DCF application.

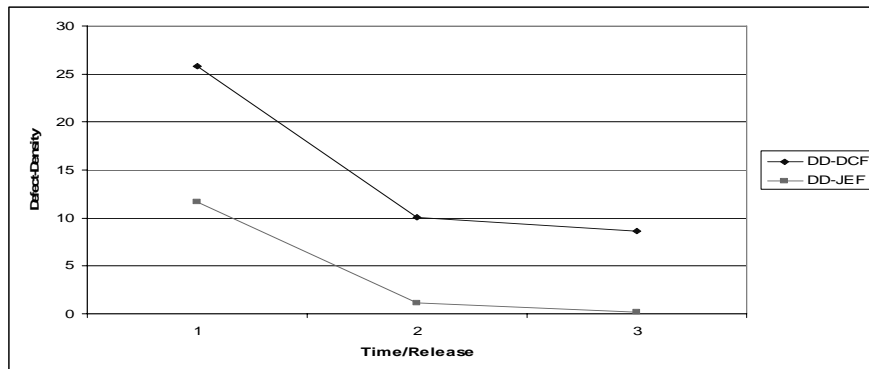


Figure 1: Defect-density per release

Table 3 gives the overview of the defect-density (#defects/KLOC) for the three releases of JEF framework and DCF application.

Table 3: Defect-density for JEF framework and DCF application

Releases	Defect-density JEF	Defect-density DCF	Percentage lower (JEF vs. DCF)
Release 1	11.67	25.84	54.83%
Release 2	1.18	10.02	88.22%
Release 3	0.20	8.62	97.67%

We can see from Table 3 that the defect-density of the JEF framework, as well as of the DCF application, decreases over releases. However, the reusable JEF framework has lower defect-density (Release 1:

54.83%, Release 2: 88.22% and Release 3: 97.67%) than the DCF application. In summary, we can support the notion that the JEF framework has lower defect-density than the DCF application over several releases.

RQ2: How does the change-density for the reusable framework vs. the application evolve over several releases? For the DCF application, 17 out of the 127 change requests for the DCF application have been related to other applications than DCF itself and are therefore excluded from our analysis. In total, we have used 205 change requests for the JEF framework, and 110 change requests for the DCF application in the analysis for RQ2. We plotted our data in a lineplot, seen in Figure 2. From this figure we can see that the change-density for the JEF framework decreases, while for the DCF application a slow increase appears.

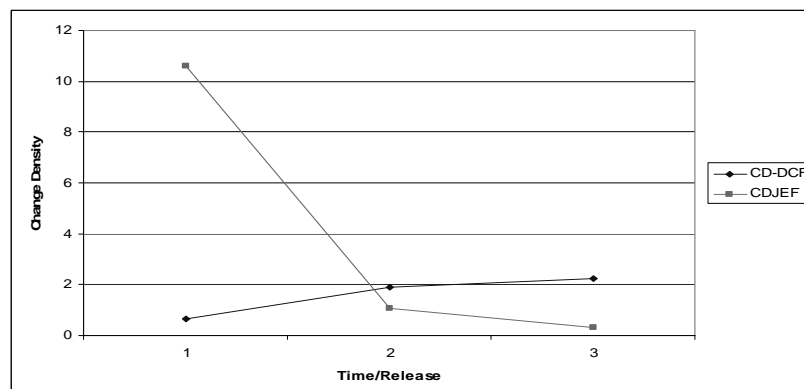


Figure 2: Change-density per release

Table 4 gives the overview of the change-density (#change requests/KLOC) for the three releases of JEF framework and DCF application.

Table 4: Change-density for JEF framework and DCF application

Releases	Change-density JEF	Change-density DCF	Percentage lower (JEF vs. DCF)
Release 1	10.61	0.63	DCF 94.06% lower than JEF
Release 2	1.08	1.91	43.45%
Release 3	0.29	2.23	86.99%

We can see from Table 4 that the change-density for the reusable JEF framework is higher (Release 1: 94.06%) in the first release, but lower (Release 2: 43.45% and Release 3: 86.99%) than the DCF application over the successive releases. The difference is significant, and hence the framework gets more

stable. In summary, we can support the notion that the JEF framework has lower change-density than the DCF application for release 2 and 3. However, for the DCF application a slow increase appears.

RQ3: What is the relation between change-density and defect-density for the reusable framework, in comparison with the application over several releases? In the analysis of RQ3, a total of 205 change requests and 223 trouble reports were used for the JEF framework, while for the DCF application a total of 110 change requests and 966 trouble reports were used. We plotted our data in a lineplot, and Figure 3 shows the change-density vs. the defect-density for the reusable JEF framework and the DCF application (rls means release). For the reusable JEF framework, we can see that the change-density and defect-density decrease over releases. This shows that the reusable components become more stable over several releases. When it comes to the DCF application, the graph shows that while defect density decreases, change-density increases.

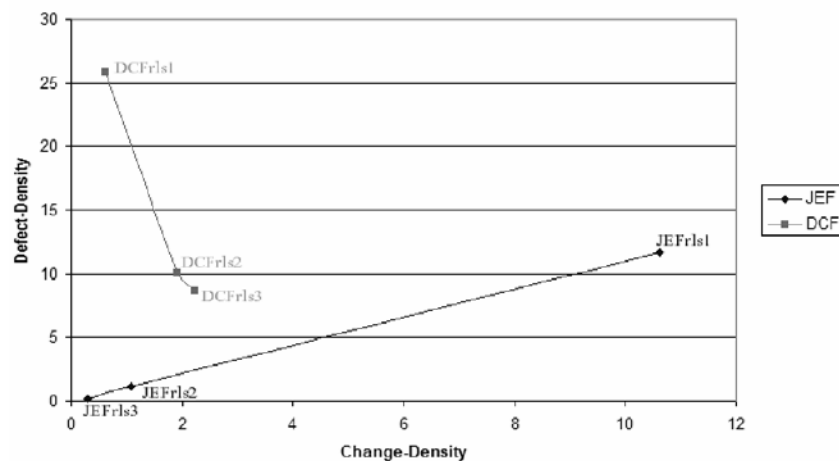


Figure 3: Change-density vs. Defect-Density

Nevertheless, Figure 3 shows that the reusable JEF framework has a lower change-density as well as a lower defect-density than the DCF application.

7. Summary and discussion

In Table 5, we have summarized our analysis results, along with the corresponding research questions.

Table 5: Summary of the results

Quality Focus	ID	Research question	Result
Defect-density	RQ1	How does the defect-density for the reusable framework vs. the application evolve over several releases?	<i>JEF framework has lower defect-density than the DCF application over several releases.</i>
Change-density	RQ2	How does the change-density for the reusable framework vs. the application evolve over several releases?	<i>JEF framework has higher change-density in the first release, but lower change-density than the DCF application over the successive releases. The difference is significant.</i>
Change-density vs. defect-density	RQ3	What is the relation between change-density and defect-density for the reusable framework, in comparison with the application?	<i>JEF framework: decreasing defect-density and change-density. DCF application: decreasing defect density while increasing change-density.</i>

7.1 RQ1: How does the defect-density for the reusable framework vs. the application evolve over several releases?

Our results show that the JEF framework has lower defect-density than the DCF application over three releases in Statoil ASA. This is likely due to the JEF framework being more mature than the DCF application. A possible cause may be that the functionality of the JEF framework is more stable, since the focus of the JEF framework is to be reused by other applications. Hence, our findings confirm previous results in literature [7]. Decreasing defect-density means fewer corrections are needed, and thereby a higher quality level is achieved for the reusable JEF framework in Statoil ASA.

Some previous literature has considered the size factor as a possible connection to defect-density [15][16][17]. We do not currently possess detailed data on the subsystems of the DCF application, but what can be seen from the size data presented here is that there is a difference in size between the DCF application and the JEF framework, with the DCF application being larger over all three releases (see Table 1 and Table 2). This could mean that DCF components are larger than JEF components in general, and may thereby partly account for the generally higher defect density seen for the DCF application.

7.2 RQ2: How does the change-density for the reusable framework vs. the application evolve over several releases?

Our findings support the results of [7]. For the DCF application we saw a slow increase in change-density, possibly due to the implementation of new and changed requirements which become more complex over time.

When it comes to the JEF framework, we can see a decreasing change-density. A possible explanation here is that the JEF framework becomes more mature, as it is being changed and adapted to accommodate new and changed requirements. That is in relation to the existing and new applications that reuse the JEF framework.

This is an important observation in the sense that it allows for a relatively stable amount of resources to be assigned towards change requests for the reusable JEF framework between releases. Also, a decreasing change-density can be used as a measure of relative stability in the requirements for the JEF framework, towards showing that they are widely applicable to the various systems in which the JEF framework currently is and in the future could be used for development.

7.3 RQ3: What is the relation between change-density and defect-density for the reusable framework, in comparison with the application?

The increase in change-density for the DCF application may indicate that more complex changes are incurred over time. It may also be caused in part by an immature reuse program – Statoil initiated their strategy in 2003 with initial prestudies, and the program itself was started in 2004. One consequence of the change-density characteristics seen is that more developer effort must be used towards implementing changes for the DCF application.

This increase is not seen for the JEF framework, which could indicate that the developers working on this team are more experienced. However, this would only be part of the explanation, as we know that some project members hold multiple responsibility roles between the JEF and DCF development teams [19],

and hence work on both teams. Another factor is that the changes that the JEF framework incurs may be fewer and less complex as the framework becomes more stable and hence better adapted to the various systems that it serves, as discussed in the previous section 7.2.

One of the possible explanations for the declining defect-densities for both the DCF application and the JEF framework is that developers gain experience over time, so that changes are easier to make and cause less defects when introduced. Also, another confounding factor to explain the declining defect-density is that changes may get less complex over time. There are factors other than changes which affect defect-density such as size, history or complexity of components, which should be further studied.

7.4 Threats to validity

Here, we discuss the possible threats to validity in our case study, using the definitions provided by [13]:

Construct Validity: The metrics we have used (defect-density and change-density) are thoroughly described and used in literature. All our data is of pre-delivery change requests and trouble reports from the development phases for the three releases of the reusable framework, and for the three releases of the DCF application.

External Validity: The entire data set is taken from one company. The object of study is a framework consisting of only seven components, and only one application. The whole data set of trouble reports and change requests in Statoil ASA has been collected for three releases of the reusable framework, as well as for three releases of the application. So, our results should be relevant and valid for other releases of the framework and other future releases of the application. Generalization to similar contexts in other organizations should be discussed by case.

Internal Validity: All of the change requests and trouble reports for the JEF framework and the DCF application have been extracted from Statoil ASA by us. Incorrect or missing data details may exist, but these are not related to our analysis of defect-density and change-density.

Conclusion Validity: This analysis is performed based on an initial collection of data. A threat to the conclusion validity is how the defects and changes are reported at Statoil ASA. Ambiguity could exist as to whether developers classify an incident as a Trouble Report or a Change Request, hence this remains a threat. Another threat could be if more experienced developers work with certain types of components.

However, this is not considered a threat to our investigation, since the JEF framework and the DCF application are developed within the same development unit and although by separate teams, with comparable skills. Even though our data set of change requests and trouble reports should be sufficient to draw relevant and valid conclusions, it is still a small size. If we had more components, more applications, as well as more releases to compare, the conclusions would be more reliable. Also, we have only studied one application, so we had to use all the data material from this application to get sufficient data. As new JEF releases, new releases of the DCF application, as well as other applications (reusing the JEF framework) are released, they should be included in our dataset in future studies to see if they support the same tendencies as discovered here.

8. Conclusion and future work

We have presented the results from a case study of defect-density and change-density of a reusable framework and one application reusing it in Statoil ASA. We have defined three research questions, and the results are presented in Table 5. There are few published empirical studies which characterize and verify reuse benefits over releases. Hence, our study is a contribution in that context. The results can also be used as a baseline to compare future studies of software reuse.

The results are presented to Statoil ASA and contribute towards understanding the maintenance and evolution of the reusable JEF framework and the DCF application. Lower change-density (higher stability) and lower defect-density of the JEF framework show the industrial advantage of reuse. The results will also be combined with other research in the company to find and explain future approaches regarding component reuse. One interesting question raised from their side is whether the results of this work can be used as input to improve future reuse programs. Additionally, we plan to expand our dataset to include future releases of the JEF framework, future releases of the DCF application, as well as new applications (reusing JEF framework), and to refine the research questions based on our initial findings here.

9. Acknowledgement

This work has been done as a part of the SEVO project (Software EVolution in component-based software engineering), an ongoing Norwegian R&D project from 2004-2008 [9], and as a part of the first

and second authors' PhD study. We would like to thank Statoil ASA for the opportunity to be involved in their reuse projects.

10. References

- [1] L. A. Belady and M. M. Lehman; *A model of a Large Program Development*, IBM Systems Journal, 15(1):225-252, 1976.
- [2] K. H. Bennett and V. Rajlich; *Software Maintenance and Evolution: A Roadmap*, ICSE'2000 – Future of Software Engineering, Limerick, Ireland, 2000, pp. 73-87.
- [3] A. Endres and D. Rombach, *A Handbook of Software and Systems Engineering*; Empirical Observations, Laws and Theories, Pearson Addison-Wesley, 2004.
- [4] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12, 1990.
- [5] IEEE Std. 1219: *Standard for Software Maintenance*, Los Alamitos, CA, USA, IEEE Computer Society Press, 1993.
- [6] W. C. Lim; *Effect of Reuse on Quality, Productivity and Economics*, IEEE Software, 11(5):23-30, Sept./Oct. 1994.
- [7] P. Mohagheghi, R. Conradi, O. M. Killi, H. Schwarz, *An Empirical Study of Software Reuse vs. Defect Density and Stability* in Proc. 26th Int'l Conference on Software Engineering (ICSE'2004), 23-28 May 2004, Edinburgh, Scotland, pp. 282-291, IEEE-CS Press Order Number P2163.
- [8] W. Selby, *Enabling reuse-based software development of large-scale systems*, IEEE Trans. SE, 31(6), pp. 495-510, June 2005.
- [9] The Software EVolution (SEVO) Project, <http://www.idi.ntnu.no/grupper/su/sevo/>
- [10] G. Sindre, R. Conradi, and E. Karlsson: *The REBOOT Approach to Software Reuse* in Journal of System Software, 30(3): 201-212, 1995.
- [11] I. Sommerville; *Software Engineering*, Sixth Edition, Addison-Wesley, 2001.
- [12] O&S Masterplan at Statoil ASA, <http://intranet.statoil.no>
- [13] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén: *Experimentation in Software Engineering – An Introduction*, Kluwer Academic Publishers, 2002, ISBN 0-7923-8682-5.
- [14] JEF Concepts and Definition at Statoil ASA, 2006, <http://intranet.statoil.com>
- [15] T. J. Ostrand and E. J. Weyuker: *The distribution of faults in a large industrial software system*, Proc. The International Symposium on Software Testing and Analysis (ISSTA-02), ACM SIGSOFT Software Engineering Notes, p. 55-64, 2002.
- [16] N. E. Fenton and N. Ohlsson, *Quantitative analysis of faults and failures in a complex software system*, IEEE Trans. Software Engineering, 26(8):797-814, 2002.
- [17] K. Y. Malaiya and J. Denton, *Module size distribution and defect density*, Proc. 11th International Symposium on Software Reliability Engineering (ISRE-00), p. 62-71, 2000.
- [18] V. R. Basili, L. C. Briand and W. L. Melo, *How reuse influences productivity in object-oriented systems*, Communications of the ACM, 39(10), October 1996.
- [19] O. P. N. Slyngstad, A. Gupta, R. Conradi, P. Mohagheghi, H. Rønneberg, E. Landre, *An Empirical Study of Developers Views on Software Reuse in Statoil ASA*, Jose Carlos Maldonado and Claes Wohlin (Eds.): Proc. 5th ACM-IEEE Int'l Symposium on Empirical Software Engineering (ISESE'06), Rio de Janeiro, 21-22 September 2006, IEEE Press, ISBN 1-59593-218-6, p. 242-251.