

An Empirical Study of Software Changes in Industry - Origin, Priority Level and Relation to Component Size¹

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

This paper describes the results of analyzing change requests from 4 releases of a set of reusable components developed by a large Oil and Gas company in Norway, Statoil ASA. These components are total 20348 SLOC (Source Lines of Code), and have been programmed in Java. Change requests in our study cover any change in the requirements. We have investigated the distribution of change requests over the categories perfective, adaptive and preventive changes that characterize aspects of software maintenance and evolution. In total there are 208 combined perfective, adaptive and preventive changes. The results reveal that 59% of changes are perfective, 27% of changes are adaptive and 14% of changes are preventive. The corrective changes (223 in total) are excluded in this paper, since they will be analyzed in future work. We have also investigated the relation between customers' and developers' priority on change requests and found no significant difference between customer and developers' priority of change requests. Larger components had more change requests as expected and priority level of change requests increases with component size. The results are important in that they characterize and explain the changes to components. This is an indication as to which components require more effort and resources in managing software changes at Statoil ASA.

1. Introduction

There have been few published, longitudinal empirical studies on industrial systems. Many organizations gather a large amount of data related to their process and products. However, the data analyses are not done properly, or the results are kept inside the organization. This paper presents the results of an empirical study of software changes, where particularly origin, priority level and relation to component size are investigated. Software changes are an important source of information for studying software maintenance and evolution. Such changes are frequent in most software systems, and are responsible for a large part of the software costs. Prior studies have investigated the aspects of maintenance, e.g. the variations in amount of maintenance activities, where these changes are located, and what the consequences of these changes are. Due to the dynamic nature of software, we need to revisit these questions and answers to ensure that the findings remain valid, and that it is possible to discover new and additional results.

¹ This paper was presented at the NIK-2006 conference. For more information, see [//www.nik.no/](http://www.nik.no/).

Currently, we are studying the reuse process in the IT-department of a large Norwegian Oil & Gas company named Statoil ASA² and collecting quantitative data on reused components. The research questions include the distribution of change requests, the relation between customer and developers' priority on change requests, the relation between component size and number of change requests, as well as understanding the relation between component size and priority level of change requests. Based on these issues, we have defined and explored several research questions and hypotheses through an empirical study.

The results support some conclusions from earlier studies, but have also discovered new and additional results. The number of change requests is to some extent small, and future studies will be used to refine and further investigate the research questions and hypotheses presented here. This paper is structured as follows: Section 2 discusses software changes, Section 3 has related work, Section 4 introduces our context Statoil ASA, and Section 5 discusses research background and motivation. Furthermore, Section 6 contains the results of our analysis of change requests, Section 7 discusses these results, while Section 8 concludes.

2. Terminology

The IEEE definition [4] for software maintenance is “*Software maintenance is the process of modifying a component after delivery to correct faults, to improve performances or other attributes, or adapt to a changed environment*”. However, many researchers believe that software maintenance starts before delivery. Software evolution does not have a standard agreed-upon definition in the literature, but some researchers use it in place of maintenance when software is also enhanced. Lastly, Belady and Lehman [1] initially defined software evolution to be “...*the dynamic behaviour of programming systems as they are maintained and enhanced over their life times.*” This definition indicates that evolution should encompass maintenance activities.

Lehman [7] carried out the first empirical work on software changes, finding that systems that operate in the real world have to be adapted continuously, otherwise, their changeability decreases rapidly. During the lifetime of software systems, they usually need to be changed as the original requirements may change to reflect changing business, user and customer needs [12]. Other changes occurring in a software system's environment may emerge from undiscovered errors during system validation, requiring repair or when new hardware is introduced. Postema et al. [12] have characterized changes in a software system to include: 1) alterations to fix coding errors, 2) more involved changes to fix design errors, and finally 3) thorough changes to fix specification errors or implement new requirements.

In general, the literature divides changes to software into four classes – namely corrective, adaptive, perfective and preventive. In general, corrective refers to fixing bugs, adaptive has to do with new environments or platforms, while implementing altered or additional new requirements, as well as improving performance, can be classified as perfective. Finally, changes made to improve future maintainability can be thought of as preventive [15]. Small differences may exist in the definition of these change classes, which can make the comparison of studies difficult. For example, Mockus and Votta [9] have classified enhancements as adaptive changes, and

² ASA stands for “allmennaksjeselskap”, meaning Incorporated.

optimizations as perfective changes. We have decided to use adaptive changes to be changes related to adapting to new platforms or environments. Perfective changes to encompass new or changed requirements as well as optimizations, while preventive changes to have to do with restructuring and reengineering. These definitions match with that of Mohagheghi and Conradi [10]. Studying the distribution is important to discover where the majority of the effort related to changes is being spent in Statoil ASA. Also, from this possible management strategies can be suggested, depending on which type(s) of changes that are more prevalent.

3. Related work

Understanding the issues within software maintenance and evolution, on a lower level involving software changes, has been a focus since the 70's. The aim has been to identify the origin of a change, as well as the frequency and cost in terms of effort. Software changes are important because they account for a major part of the costs of the software. At the same time, they're necessary; the ability to alter software quickly and reliably means that new business opportunities can be taken advantage of, and that businesses thereby can remain competitive [2].

Lientz, Swanson & Tompkins [8] did a study, where they surveyed 69 system and department managers from as many organizations on maintenance effort and activities. Overall, they found the distribution of the maintenance activity to be 18.2 % adaptive, 17.4% corrective, 60.3% perfective and 4.1% other. Other results include that maintenance activities consume much of the available resources, and that maintenance is viewed as more important than development of new application. Also, changes originating from the customer contribute to the most important area of focus for the managers.

Schach et al. [13] did a case study of 3 software products (a commercial real-time product, a Linux kernel, and GCC), based on the work of Lientz et al. [8]. However, Schach et al. found significantly different results; 4.4 % adaptive, 53.4 % corrective, 36.4 % perfective and 0.0 % other, when looking at the code modules. A similar distribution was found when considering the change logs; 2.2% adaptive, 56.7% corrective, 39.0% perfective and 2.4% other. The category "other" here encompasses preventive maintenance.

Another interesting case study is done by Lee & Jefferson [6]. Their study shows the maintenance distribution of a Web-based java application, consisting of 239 classes and 127 JSP files. This study reveals that the distribution of effort over maintenance activity categories as 32 % corrective, and 68% combined perfective, adaptive and preventive, claiming similarity with previous maintenance studies.

The aforementioned studies include corrective changes, which we are not looking at in this study. Therefore, our results will not be directly comparable to these studies. They are nevertheless important to see the general trends in software change distributions, as well as to see why changes occur in software. Mohagheghi and Conradi [10] have done an empirical study of change requests in 4 releases of a large-scale telecom system. Their focus has been on the origin, acceptance rate and functionality vs. quality attributes of the software changes. Their study reveals that previous releases of the system are no longer evolved and perfective changes to functionality and quality attributes are most common. For each release, the functionality is enhanced and

improved. When it comes to the quality attributes they are mostly improved and have fewer changes related to new requirements. Hence, the adaptive/preventive changes are lower, but not as low as reported in some previous studies. The maintenance distribution in their study is; 61% perfective, 19% adaptive, 16% preventive and 4% other (“other” here refers to saving money/effort). We will compare our results to this study since they have used the same change categories as us.

4. Statoil ASA

Statoil ASA is a large, multinational company, in the oil & gas industry. It is represented in 28 countries, has a total of about 24,000 employees, and is headquartered in Europe. The central IT-department in the company is responsible for developing and delivering software, which is meant to give key business areas better flexibility in their operation. They are also responsible for operation and support of IT-systems at Statoil ASA. This department consists of approximately 100 developers worldwide, located mainly in Norway and Sweden. 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 [5]. Statoil ASA has chosen to call this customized framework for the “JEF framework”.

This IT strategy was started as a response to the changing business and market trends, and in order to provide a consistent and resilient technical platform for development and integration [11]. The strategy is now being propagated to other divisions within Statoil ASA. Adapted to the purpose and context of Statoil ASA [11], the JEF framework itself consists of seven different components, namely JEF Client (8885 LOC), JEF Workbench (4748 LOC), JEF Util (1647 LOC), JEF Integration (958 LOC), JEF SessionManagement (1468 LOC), JEF Security (2374 LOC) and finally JEF DataAccess (268 LOC). These JEF components are total 20348 SLOC (Source Lines of Code) in size, and can either be applied separately or together when developing applications. In this paper, we will be studying the JEF components.

4.1. Change Request data in Statoil ASA

Statoil ASA cooperates with the SEVO (Software EVolution) project [14] and has given us access to data for analysis and feedback. There are two types of changes defined by Statoil ASA [16], namely; (1) *scope changes* which are (Change Requests – CR) related to perfective, adaptive and preventive changes, and (2) *incidents* (Trouble Reports - TR) which are identified as defects and errors, related to corrective changes, that leads to wrong performance of the system and need to be corrected. In this paper, we are focusing exclusively on scope changes (defined by us as change requests, since it denotes the same thing). These change requests are the source of changes between releases, following the first release. This means that the change requests show evolution between releases and on the whole. We are hence looking at perfective, adaptive and preventive changes, excluding corrective changes since they will be analyzed by us in another paper.

When a change request is identified, it is written and registered in Rational ClearQuest. Examples of change requests are:

- add, modify or delete functionalities (perfective maintenance)
- solve an anticipated problem (preventive maintenance)

- adapt to changes from other JEF component interfaces (adaptive maintenance)

Statoil ASA has a change request workflow regarding registration and implementation of changes requests [16]. A change request may impact one or more of the seven JEF components, but will usually impact only one of them. If a change request impacts several components, it will be related to the category *General*. This is due to that these change requests impact the JEF framework as a whole, and hence cannot be assigned to one or several of the components while excluding others. Each change request contains an ID, headline description, priority given by both customer and developer (Critical³, High⁴, Medium⁵ or Low⁶), estimated time to fix, remaining time to fix, subsystem location (one of the seven JEF components), system location (JEF), as well as an updated action and timestamp record for each new state the change request enters in the workflow [16].

Change requests are registered in Rational ClearQuest tool and are exported to Microsoft Excel for analysis. The first change request data for the JEF components were obtained on October/November 2005, and contained 204 change requests. An updated version of the change request data were later obtained on February 2006, and consisted of 208 change requests. The change requests are from the 4 releases of the JEF components: release 2.9 was released 14.06.2005, release 3.0 was released 09.09.2005, release 3.1 was released 18.11.2005, and finally release 3.2 is still under development. All JEF releases prior to 2.9 have been for internal development only, while release 2.9 is the first to be reused in other development projects at Statoil ASA. The change requests were then classified manually according to the software change classification (adaptive, perfective and preventive) by the first and second author together, see Section 2 of this paper for the definitions used.

5. Research method, questions and hypotheses

Our motivation is to investigate software changes and how they are handled in Statoil ASA. In this section, our research questions, method and hypotheses are presented.

In our study we decided to refine our research questions with corresponding hypotheses. A hypothesis is stated formally, and believed to be true about the relation between one or more attributes of the *object of study*, and the *quality focus* [10]. The process of choosing research questions and hypotheses for our study has been done by using a combined top-down and bottom-up process. RQ1 was selected from the literature (top-down), while the remaining ones were chosen after a pre-analysis on the data collected from ClearQuest (bottom-up). The following is a presentation of our research questions and hypotheses.

RQ1: How is the distribution of change requests over perfective, adaptive and preventive changes? A study of RQ1 is important since it gives us the possibility to see why changes occur in software. Some prior research [10] has found that perfective and adaptive changes account for the majority of the changes, while preventive follows

³ A CR with this grade means that the system does not fulfill critical business functionality or will disrupt other systems.

⁴ A CR with this grade means loss of a part of the required functionality or quality.

⁵ A CR with this grade also means loss of a part of the required functionality or quality, but there exists ways to work around the problem.

⁶ A CR with this grade is has no importance on the functionality and quality.

closely. The purpose here is to confirm whether these change types can be seen with a similar percentage distribution in Statoil ASA.

RQ2: What is the relation between the customer priority and the developers' priority on change requests? A study of RQ2 is important since we get the chance to see whether customer and developers' have the same perception of priority given for change requests. At the inception of a change request, a priority level is given by the customer. Developers also set a priority on each change request once submitted in ClearQuest. Here, we want to study the consistency between the customer and developers' priority, to determine whether there is a tendency towards higher or lower priority when comparing these against each other. The following are the related hypotheses for RQ2:

H02: There is no difference in the prioritizing of change requests by customers and developers.

HA2: Developers give a higher priority than customers to the change requests.

HB2: Developers give a lower priority than customers to the change requests.

RQ3: What is the relation between component size and the number of change requests? A common belief among developers in Statoil ASA is that larger components face more changes. We have not found any literature supporting or declining this belief. Here, we want to check whether this belief among the developers in Statoil ASA is true or not. The following are the related hypotheses for RQ3:

H03: There is no relation between component size in SLOC and the number of change requests.

HA3: The number of change requests increases with the component size in SLOC.

RQ4: What is the distribution of change requests over priority levels given by developers? Another common belief among developers in Statoil ASA is that larger components face more serious (critical and high) change requests. Here too, we have not found any literature supporting or declining this belief. Therefore, we want to check whether this belief among the developers in Statoil ASA is true or not. Specifically, we are interested in finding out whether the larger components have more critical and high change requests than the smaller ones.

6. Results of the Study

The statistical analysis tools we used were SPSS version 14.0 and Microsoft Excel 2003. In our analysis we will conclude with a significance level (α -value) of 0.05 or below for each research questions individually. We will also report the observed significance level (p-values). In total, there are 208 change requests spanning all four releases, and we have used all these 208 change requests in our analysis unless otherwise specified below.

6.1 RQ1: How is the distribution of change requests over perfective, adaptive and preventive changes? Our result for RQ1 shows that the distribution of change requests over the following categories is as follows: 59% perfective, 27% adaptive and 14% preventive.

6.2 RQ2: What is the relation between the customer priority and the developers' priority on change requests? For RQ2 we decided to use Paired T-test [3]. 27 out of 208 change requests have not been given any priority by the customer *and/or* the developers, and these 27 change requests are not included in the analysis here. In total,

we have used 181 remaining change requests in the analysis of RQ2. The data in Figure 1 shows that some differences do exist; customers have set “critical” priority on more change requests than developers, while developers have assigned more “high” priority to change requests. The number of change requests with priority level “medium” and “low” are the same for both developer and customer. From the figure, we may conclude that there are no significant differences in prioritizing, except for some difference in deciding what is critical.

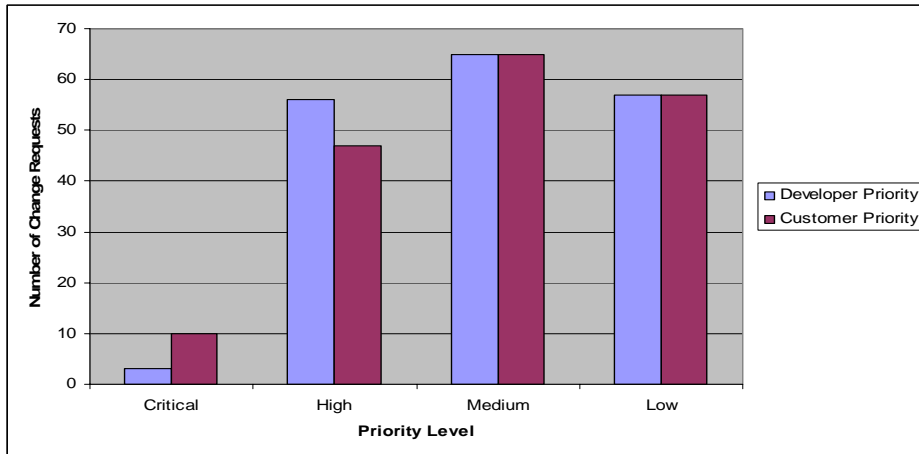


Figure 1: Priority level of developer vs. customer

With the Paired T-test, we wanted to see if there was a significant difference in the mean-values of the priority given to the change requests. The significance level is 0.05, and the data were checked for normality. The Paired T-test we performed yielded a t-value of 0.882, degree of freedom = 180 and the critical value = 1.960. Since $0.882 < 1.960$, it is not possible to reject H_0 .

6.3 RQ3: What is the relation between component size in SLOC and the number of change requests? For RQ3 we decided to use a Spearman’s rho correlation [3]. First we plotted our data in a scatterplot, seen in Figure 2. From this figure, we can clearly see that number of change requests increases with the component size, though not strictly linearly.

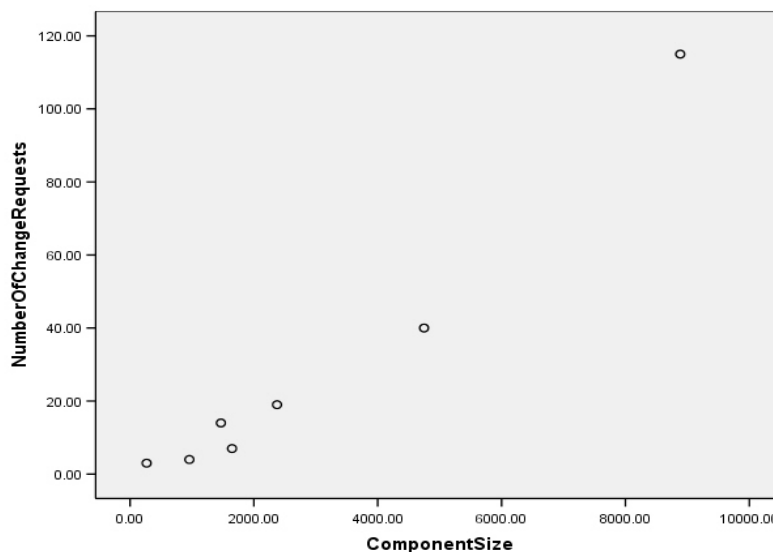


Figure 2: Component size in SLOC vs. Number of change requests

6 out of the 208 change requests we analyzed have been associated with the category *General* (see section 4.1). These 6 change requests have therefore been excluded from our analysis here. In total, we have used 202 remaining change requests in the analysis of RQ3. The Spearman’s rho correlation we performed yielded a Correlation coefficient = 0.964, Significance (1-tailed test) = 0.000 and N = 7, with correlation significant at the 0.01 level (p-value). Since $0.000 < 0.01$, it is possible to reject the null hypothesis at the level of p-value. In summary, we can reject H03 in favour of our alternative hypothesis HA3, and hence support the notion that the number of change requests increases with component size. Since the number of components is low, we cannot solely rely on statistics here, but we also observe a clear relation from Figure 2.

6.4 RQ4: What is the distribution of change requests over priority levels given by developers? For RQ4 we have excluded 13 change requests which have not been given any priority, as well as 6 change requests that fall into the category *General* (see section 4.1). We have used a total of 189 change requests in the analysis of RQ4. We plotted our data in a histogram, seen in Figure 3.

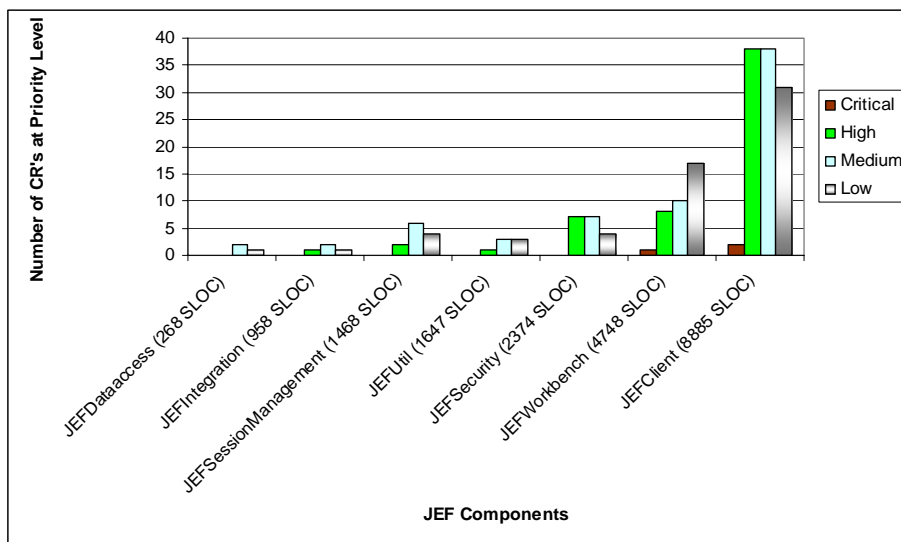


Figure 3: Distribution of priority level given by developers’ pr. JEF component

From this figure, we can see that the larger components have more critical and high priority on change requests (given by developers). However, JEF Util is slightly larger than JEF SessionManagement (the difference is 179 SLOC), but still has slightly fewer CRs ranked high. Finally, we see from the Figure 3 that the smaller components do not have critical CRs. In particular, JEF Client (largest component) has the following distribution: 1.8% (2/109 of CRs) critical, 34.9% (38/109 of CRs) high, 34.9% (38/109 of CRs) medium, and 28.4% (31/109 of CRs) low. In comparison, JEF Dataaccess (smallest component) has the following distribution: 0.0% critical, 0.0% high, 66.6% (2/3 of CRs) medium, and 33.3% (1/3 of CRs) low. This shows that, in general, the larger components have more critical and high change requests than the smaller ones.

Overall, our results can be summarized as follows in Table 1.

Table 1: Summary of the results

Research Questions	Hypotheses	Results
RQ1	<i>None.</i>	The distribution of change requests are as follows: 59% perfective, 27% adaptive and 14% preventive.
RQ2	<i>H02: There is no variation in the prioritizing of change requests. HA2: Developers give a higher priority than customers on change requests. HB2: Developers give a lower priority than customers on change requests.</i>	Not rejected Not rejected Not rejected
RQ3	<i>H03: There is no relation between component size and number of change requests. HA3: The number of change requests increases with component size.</i>	Rejected Not rejected
RQ4	<i>None.</i>	The distribution of priority levels on change requests for JEF Client: 1.8% (2/109 of CRs) critical, 34.9% (38/109 of CRs) high, 34.9% (38/109 of CRs) medium, and 28.4% (31/109 of CRs) low. In comparison, JEF Dataaccess has the following distribution: 0.0% critical, 0.0% high, 66.6% (2/3 of CRs) medium, and 33.3% (1/3 of CRs) low.

7. Discussion

7.1 RQ1: How is the distribution of change requests over perfective, adaptive and preventive changes? Our results confirm some of the findings from an earlier study [10]. This result means that the majority of developers' effort for changes to JEF components is related to perfective changes (e.g. new or changed requirements as well as optimizations), closely followed by adaptive changes. Only a smaller portion of this effort is spent on preventive changes.

7.2 RQ2: What is the relation between the customer priority and the developers' priority on change requests? On the relation between the priority assigned by the customer and the corresponding priority given by the developers on change requests, we have not seen significant differences. However, the data shows that there is a difference for critical change requests, though not statistically significant. The data trend is that the customer assigns more change requests on the *critical* level than the developers, while developers assign more on the *high* level than customers. This is due to developers having downgraded critical ones to high priority. Also, an inherent psychological effect may exist, as the developers actually have to make the changes, while the customers do not.

7.3 RQ3: What is the relation between component size and number of change requests? Our results indicate that the larger the components, the more changes they incur. This may not be a surprising result, but verifying this was important to Statoil ASA in order to show where the majority of change requests occur. This may introduce a problem for maintainability of large components, and stresses that these components should be designed to facilitate change. We also attempted to further our investigation with looking at the change-density (defined as the number of change requests divided by Source Lines of Code), but this only yielded a diagram of completely scattered

datapoints. This problem is caused by the relatively large difference in size between the two larger components, and the remaining smaller ones.

7.4 RQ4: What is the distribution of change requests over priority levels given by developers? We have found that the larger components have more serious (critical and high) change requests than the smaller ones, and that the smaller components do not have critical change requests at all. The difference between the components JEF Util and JEF SessionManagement seen in the results can be accounted for by the small difference in size between the two components. The result of RQ4 is interesting, and may have its origin in that larger components incur more changes hence it is of a higher priority to get these changes put in place. It may also be that larger components simply are those that incur more serious changes. Also, our results match with the developers' expressed opinion that JEF Client is more complex to develop and maintain, as this component incurs the most changes with the higher priority levels.

7.5 Threats to validity

We here discuss the possible threats to validity in our study, using the definitions provided by Wohlin [17]:

Construct Validity: All of the data categories we have used (perfective, adaptive and preventive) have been extracted from and are well-founded concepts in the software evolution and maintenance field. All our data are of pre-delivery change requests from the development phases. This is similar to at least one study [10].

External Validity: The entire data set is taken from one company. The object of study is a framework consisting of only seven components, and the data has been collected for 4 releases of these components. Our results should be relevant and valid for other releases of these components, as well as for similar context in other organizations.

Internal Validity: All of the change requests for the JEF components have been classified manually by us. We have performed the classification separately, and then compared the results jointly. Nevertheless, this can have lead to wrong classifications in a few cases.

Conclusion Validity: This analysis is performed based on an initial collection of data. Even though, this data set of change requests should be sufficient to draw relevant and valid conclusions, it is still a small size. As new JEF releases are released, they should be included in our dataset to see if they support the same tendency as discovered here.

8. Conclusion and future work

We have presented the results of the empirical study of change requests in Statoil ASA on origin, priority level and relation to component size. We have defined 4 research questions and deployed two of them in hypotheses. We don't claim that our results are surprising. However, there are few published empirical studies which characterize and explain the changes to software components, in this case reusable ones. Hence, our study is a contribution in that context. The results can also be used as a baseline to compare future studies of software changes, and to compare the results to changes of non-reusable components.

The results have been presented to Statoil ASA and contribute towards understanding the origin of changes and the criticality of them. All these insights represent explicit knowledge, and will be important for deciding how to manage future software changes

in the company. The results will also be combined with other research in the company to further explain our findings. One interesting question raised from the company 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 corrective changes, and to refine the research questions based on our initial findings here. Concretely, one direction will be to see how the level of priority assignment changes over time; i.e. whether more critical changes will be made in future or changes will be of less impact.

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 [14], 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] Donald Cooper and Pamela Schindler; *Business Research Methods*, 8th Edition, McGraw-Hill, 2003, ISBN 0-07-115160-5
- [4] IEEE Std. 1219: *Standard for Software Maintenance*, Los Alamitos, CA, USA, IEEE Computer Society Press, 1993.
- [5] JEF Concepts and Definition at Statoil ASA, 2006, <http://intranet.statoil.com>
- [6] M. G. Lee and T. L. Jefferson; *An Empirical Study of Software Maintenance of a Web-based Java Application*, Proceedings of the IEEE International Conference on Software Maintenance, ICSM 2005, 25-30 September, Budapest, Hungary, pp. 571-576.
- [7] M. M. Lehman; *Programs, Life Cycles and Laws of Software Evolution*, Proc. Special Issue Software Eng., IEEE CS Press, 68(9):1060-1076, 1980.
- [8] B. P. Lientz, E. B. Swanson and G. E. Tompkins; *Characteristics of Application Software Maintenance*, Communications of the ACM, 21(6): 466-471, 1978.
- [9] A. Mockus, L. G. Votta; *Identifying Reasons for Software Changes Using Historical Databases*, Proc. IEEE Int. Conference on Software Maintenance (ICSM'00), 2000, pp. 120-130, IEEE CS Press.
- [10] Parastoo Mohagheghi and Reidar Conradi; *An Empirical Study of Software Change: Origin, Impact, and Functional vs. Non-Functional Requirements*, In Proc. Int'l Symposium on Empirical Software Engineering (ISESE'04), 19-20 Aug. 2004, Redondo Beach, Los Angeles, USA, IEEE CS Press, ISBN 0-7695-2165-7, 296 pages, p. 7-16.
- [11] O&S Masterplan at Statoil ASA, 2006, <http://intranet.statoil.com>
- [12] M. Postema, J. Miller and M. Dick; *Including Practical Software Evolution in Software Engineering Education*, IEEE Press, 2001.
- [13] S. R. Schach, B. Jin, L. Yu, G. Z. Heller and J. Offutt; *Determining the Distribution of Maintenance Categories: Survey versus Management*, Empirical Software Engineering 8, December 2003, pp. 351-366.
- [14] The Software EVolution (SEVO) Project, 2004-2008, <http://www.idi.ntnu.no/grupper/su/sevo/>
- [15] I. Sommerville; *Software Engineering*, Sixth Edition, Addison-Wesley, 2001.
- [16] Configuration Management (CM) Practices at Statoil ASA, 2006, <http://intranet.statoil.com>

[17] 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.