Towards Understanding Large-Scale Adaptive Changes from Version Histories

Omar Meqdadi[1], Nouh Alhindawi[1], Michael L. Collard[2], Jonathan I. Maletic[1]

[1]Department of Computer Science
Kent State University
Kent, Ohio, USA
{omeqdadi, nalhinda, jmaletic}@kent.edu

[2]Department of Computer Science
The University of Akron
Akron, Ohio, USA
collard@uakron.edu

***Abstract*—A case study of three open source systems undergoing large adaptive maintenance tasks is presented. The adaptive maintenance task involves migrating each system to a new version of a third party API. The changes to support the migration were spread out over multiple years for each system. The first two systems are both part of KDE, namely KOffice and Extragear/graphics. The adaptive maintenance task, for both systems, involves migrating to a new version of Qt. The third system is OpenSceneGraph that underwent a migration to a new version of OpenGL. The case study involves sifting through tens of thousands of commits to identify only those commits involved in the specific adaptive maintenance task. The object is to develop a data set that will be used for developing automated methods to identify/characterize adaptive maintenance commits.**

***Keywords*—Adaptive Maintenance, Commit Types, Maintenance Classification.**

I. INTRODUCTION

Today's software systems have great longevity. We continually evolve and maintain them to assure their relevance and usefulness to their respective user communities. One direct side effect of longevity is that dependent platforms, compilers, libraries, frameworks, and APIs also change. Maintenance to address such changing dependent systems is termed adaptive. Adaptive-maintenance tasks involve changing a software system in response to changes in its environment. Examples include migrating a system to work on a new version of an operating system or to support a new API or library.

Adaptive maintenance is somewhat unique in that its cause is most often outside of the control of the organization/developer. That is, hardware changes, new versions of the compiler, and changes to APIs are enhancements that come from a third party. Additionally, adaptive-maintenance tasks are typically enterprise/system wide and impact large bodies of source code.

For example, many companies have recently (in the past couple years) migrated from Microsoft .Net 1.1/2.0 to .Net 3.5/4.0. This adaptive task has impacted thousands of organizations and millions of lines of code. The cost of this change is easily in the tens of millions of dollars.

Unfortunately, there is little (automated) tool support for adaptive maintenance tasks and the vast bulk of the work is done manually, thus adding to its overall cost. One of the authors' long-term research goals is to provide more automated support for the adaptive-maintenance process.

We feel one of the first steps to achieve this goal is to better understand what adaptive changes look like and how they are applied in actual software systems. There are no large studies that examine adaptive maintenance in any detailed or systematic manner. As such we undertook a case study of three open-source systems that previously underwent major adaptive maintenance tasks. We examined multiple years of version history of these systems to determine exactly which commits were involved in the particular adaptive-maintenance task. This inspection was done manually and involved reading system documentation, development notes, commit-log messages, and source code. It was a labor-intensive study that involved many person hours of work.

The result of the study was a categorization of all the commits (during a specific time frame) into either one of two categories, namely adaptive commits and non-adaptive commits. The adaptive commits involved changes to the system associated with the adaptive-maintenance task being examined. The non-adaptive commits were all other commits to the system that involved different maintenance tasks. All three of these systems continually underwent corrective maintenance and enhancements during the time of the adaptive-maintenance task. We then analyzed these adaptive-maintenance commits to identify any trends.

The remainder of this paper is organized as follows. First we describe the three systems used in the case studies and the reasons for selecting these systems. Next we present our findings from the manual investigation and describe the adaptive commits uncovered. Then an analysis of this data is presented along the measure of change size. Following is threats to validity, related work, and conclusions.

II. CASE STUDY

The goal of this case study is simple. Examine the version history of a software system and determine which commits to the system are concerned with a specific adaptive-maintenance task. We selected three open-source systems that were known to have undergone a major adaptive change in their history. Knowing the adaptive-maintenance task allowed us to narrow down the time frame of version history to examine for each system.

We now present the details of the systems along with the reasons behind the choice of each and a brief background on version-control systems and the commit process. Lastly, we present the adaptive commits that were collected as an input for adaptive change analysis.

A. Subject Systems

As a case study, we investigated and studied three large open source systems, the office-applications suite *KDE/KOffice*, graphical applications associated with the KDE project[1] in the package *KDE/Extragear/graphics*, and the high-performance 3D graphics toolkit *OpenSceneGraph* (OSG)[2]. These systems cover a number of application domains and sizes, and are prime examples of successful open-source development that utilizes a number of different APIs. Many of these third party APIs are regularly updated thus requiring migration to new API versions.

The two KDE packages we examined use Qt for the windowing and user interface. Qt is an open-source framework for developing graphical user interfaces and is widely used by many software projects. Qt uses standard C++ but makes broad use of the Meta Object Compiler, a special code generator, along with numerous macros to enhance C++.

The time frame selected for both KOffice and Extragear/graphics was from January 1, 2006 through December 31, 2010. In 2004 a major new version of Qt, namely Qt4, was released[3], and in early 2006, the developers of the KDE project started to initiate the porting of all KDE systems to support Qt4 (i.e., move from Qt 3.x to Qt 4.0). The process of moving KDE completely to Qt4 lasted until the end of 2010, as indicated by developer commits. This included dealing with another substantial release of Qt, such as Qt 4.7. That is, we are looking for adaptive maintenance related to the migration of these two KDE packages from Qt3 to Qt4.

The third system, OSG, is written in C++ using the OpenGL specification. OpenGL[4] is a software interface to graphics hardware, which offers an abstract API for drawing 2D and 3D graphics and is widely used by a number of scientific visualization and modeling systems. In August 2008, a major new version of OpenGL, namely OpenGL 3, was released. The adaptive maintenance related to the migration to OpenGL 3 was completed at the end of 2010, as mentioned in the commit annotations by the OSG developers. Hence, we investigated the commits of OSG in the time period of August 1, 2008 through Dec. 31, 2010.

B. Data Collection

The text message of a commit describes the change undertaken by the developers and the intended purpose of the change [2]. Therefore, with the text of a commit-log message we can, in most cases, determine if that commit is involved in an adaptive-maintenance task.

For the case study, we extracted the change-log file associated with each studied system using the *svn log* command. Then, we manually read and investigated each commit message in the change-log file to determine if changes that occurred in the commit were adaptive changes to the specific APIs modifications we are investigating. Here, the adaptive changes involved porting from Qt 3.x to support Qt 4.x. The adaptive modifications are identified by searching the commit-log messages for indications that Qt3 features/interfaces were changed to support new features/interfaces found in Qt4. For instance, since the class QButton in Qt3 was replaced by the class QAbstractButton in Qt4, searching for these classes in the commit-log messages is a criterion indicating adaptive commits associated with this task. Lists of changing features and classes from Qt3 to Qt4 are listed in the system's online documentation. We used the same basic process to locate the relevant adaptive changes in OSG for the migration to OpenGL 3.

Manually identifying adaptive commits was a very costly task. For instance, during the time period of January 1, 2006 to December 31, 2010 there were over 38,000 commits (in subversion) to the KOffice project. Hence, *the manual investigation took several months* of arduous work to accurately differentiate the adaptive commits from non-adaptive.

C. Summary of Findings

Our investigation shows that the vast majority of the commits during the studied time period had nothing to do with the API migrations. The majority of commits addressed other type of maintenance such as corrective, adding new features, or enhancements tasks going on in parallel. A summary of the obtained results is given in TABLE I.

It was a bit surprising how few actual commits were involved in these major API migrations. However, as we will show in the next section, the commits were on average quite large, compared to a typical commit [1], and each impacted a large number of files. Typically, a single adaptive commit involved addressing one part of the migration for the entire software system. That is, the migration process was done incrementally but system wide. The committer (developer) normally grouped the same types of adaptive transformations into a single large commit. Additionally, adaptive maintenance tasks normally do not involve large complex changes to functionality so often require few specific changes but applied in many locations.

TABLE II shows that the ratio of commits involved with these specific adaptive maintenance tasks decreased over time. Note that the OSG migration did not start until 2008. This further supports that the systems were incrementally migrated to new API versions, as the API versions were released, e.g., Qt 4.1, Qt 4.2 etc.

III. Characterizing the Adaptive Commits

We now take a closer look at the three sets of adaptive commits and attempt to uncover any similar characteristics and trends. First the commits are categorized based on how

[1] See http://www.KDE.org.
[2] See www.openscenegraph.org.
[3] See http://doc.qt.nokia.com/4.0/porting4.html.
[4] See http://www.opengl.org/registry/#oldspecs.

they impact the system. That is, what was added, deleted, or modified in each commit.

TABLE I. ADAPTIVE AND NON-ADAPTIVE COMMITS FOR THE THREE SYSTEMS OVER THE GIVEN TIME PERIOD.

System	KOffice	Extragear/ graphics	OSG
Adaptive Maintenance Task	Migration from Qt3 to Qt4	Migration from Qt3 to Qt4	Migration to OpenGL 3
Adaptive Task Starting-Date	03/29/2006	11/07/2006	09/18/2008
Adaptive Task Ending-Date	12/31/2010	12/31/2010	12/31/2010
Total Number of Commits	38,980	26,336	4,310
Number of Non-Adaptive Commits	38,849 (99.7%)	26,117 (99.2%)	4,231 (98.2%)
Number of Adaptive Commits	131 (0.3%)	219 (0.8%)	79 (1.8%)

TABLE II. DISTRIBUTION OF ADAPTIVE COMMITS PER YEAR. PERCENTAGES ARE COMPARED TO TOTAL ADAPTIVE COMMITS

	Ratio of Adaptive Commits Per Year		
Year	***KOffice***	***Extragear/graphics***	***OSG***
2006	40.5%	3.4%	0.0%
2007	24.2%	37.6%	0.0%
2008	14.1%	32.3%	50.3%
2009	11.8%	21.8%	37.1%
2010	9.4%	4.9%	12.6%

This size characteristic was selected for a variety of reasons. Given the raw data (version information) this measure is readily available from the log entries. There is little else that one can compute from the commits beyond this information. It has also been demonstrated [1] that the commit size and log message are correlated. Mockus and Votta [4] also discovered that the change size is essential to understanding why that change was performed. Hindle et al. [2] showed that the change size may be significant for predicting the type of a change.

To examine the size of the adaptive commits in the context of how many items were added, delete, or modified within each commit, the following three size-based measures were used:

- *Method*: Number of methods/functions added, deleted, or modified.
- *File*: Number of files added, deleted, or modified.
- *Module*: Number of directories that contain a change in the commit.

These measures are used to determine the overall impact a given commit has to a system. That is, a given commit (change) can be very localized to one file or function. Alternatively, it can be system wide impacting a large number of files. The goal is to characterize what a typical adaptive commit looks like. Or, more importantly, the goal is to see if there is even a typical adaptive commit. The file and module size measures are inexpensive to calculate, as only the log entries need to be analyzed. For each commit, the three size measures are computed. We use the values of these measures as the data points for classification. To aid such classification, a descriptive-statistics method was used to classify the commits into different categories based on the number of files, methods, and modules being changed. We use a 5-point summary approach, the same approach that was used for commit categorization in [1]. This categorization can then be displayed using a boxplot. We developed a tool to apply this categorization over the collected commits, both adaptive and non-adaptive, from the version history of the examined systems in the study. Here, we examined the following questions.

- What is the distribution of adaptive-change commits?
- Do adaptive-change commits cause more files to change than other commit types?
- Can we classify the adaptive-change tasks as a system-wide maintenance operation?

TABLE III provides a summary (the data, quartiles) of the commit categorization using the file-size measure for KOffice. The *Range* column of this table represents the boundaries of each defined region for the corresponding boxplot. The results show that the majority of the non-adaptive commits fit into the small and extra small categories of the examined systems. However, larger commits (large and extra-large categories) occur with a higher percentage for adaptive commits. For instance, about 21% of adaptive commits, compared with 9% of non-adaptive commits, are classified as larger commits for KOffice. Given this trend, the adaptive-maintenance tasks on average touch more files than non-adaptive tasks. That is, developers performed an adaptive task by adding/deleting/modifying code statements across the entire system, and then committed these changes using one large commit. Therefore, the file-size measurement provides an explanation as to why the number of adaptive commits is much smaller then the number of other non-adaptive maintenance commits.

In an attempt to get a more detailed picture, we further processed the files to the granularity of methods/functions. We use the GNU Unix *diff* utility to identify modified, added, and deleted methods/functions occurring in each commit of the examined systems. Our observation, for the studied systems, is that nearly 27% of the adaptive commits are in the large or extra-large categories, while only approximately 7% of the non-adaptive commits are large or extra-large.

The outcomes of commit categorization histograms using module size measurements show that the module-based results reflect the fact that the adaptive changes were often system wide. That is, changing the GUI framework impacted the entire system in a similar manner.

Moreover, we computed the level of significance of these measures using the *Mann-Whitney U* non-parametric tests. In this statistical testing, a *p-value* of 0.05 represents the significance level. If the p-value is greater than 0.05, then

we assume there is no observable difference between adaptive and non-adaptive. The results of this test for the file measure are 0.0347 for KOffice, 0.0393 for Extragear, and 0.0414 for OSG (i.e., significant). Additionally, the p-values are also significant for module and method measures.

TABLE III. FIVE-POINT SUMMARY OF COMMITS CATEGORIZED BY FILE-SIZE FOR KOFFICE.

	KOffice		
Quartile	**Number of Files**		
Q0 *(Min)*	1		
Q1	1		
Q2 *(Median)*	2		
Q3	4		
Q4 *(Max)*	3806		
IQR	Q3-Q1= 3		
Boxplot	**Range (#files)**	**Ratio in Adaptive commits**	**Ratio in Non-Adaptive commits**
Extra-Small	1-1	24.11 %	57.33 %
Small	2-4	32.59 %	23.30 %
Medium	5-8	21.69 %	10.57 %
Large	9-12	9.40 %	4.92 %
Extra-Large	>=13	12.21 %	3.88 %

IV. THREATS TO VALIDITY

Several threats to validity may influence the results of our work and the capability to generalize the obtained results. In other words, the initial research and findings seem promising, but a number of important questions still need to be addressed. The presented case study is based on investigating the version history. The packages that we examined are prime examples of successful open-source development that involve a number of API modifications. Moreover, the test systems for the preliminary experiments were chosen because of their availability and the fact that other researchers ran experiments using these systems. However, we do not claim that the obtained results would generalize to a broad range of systems, such as closed-source systems.

V. RELATED WORK

A number of studies based on software maintenance categorization have been suggested. In terms of the frequency of maintenance commits, Lientz et al. [3] published a survey of software maintenance, where they found that 18.2% of the accomplished maintenance was categorized as adaptive.

Several researchers have proposed studying and extracting information from the commits of software version histories. Robles et al. [5] investigated version histories to study the evolution of a software distribution in terms of the number of packages, lines of code, use of programming languages, and sizes of packages/files. The characterization of a typical version history commit is investigated with respect to the number of files, number of lines, and number of hunks committed together in [1]. Mockus and Votta [4] found several results on using version-control data in detecting the relationships between the type and size of the software changes. One of their results is using the difficulty, size, and time intervals to identify the type of change. Hindle et al. [2] proposed an automated classifier for large commits by training machine learners on features extracted from the commit metadata. Their results demonstrate that the change size provides helpful information to allow large commits to be classified automatically.

To the best of our knowledge, this is the first work that not only uncovers but also further analyzes adaptive change commits, with comparison to non-adaptive maintenance tasks. There is no previous work in the literature that identifies the main characteristics of adaptive commits in large-scale systems, including the commit size.

VI. CONCLUSION AND FUTURE WORK

The results of a case study to identify and analyze the adaptive changes occurring due to a migration to a new API were presented. The study is based on data obtained by manually examining the version history to distinguish adaptive from non-adaptive commits of the KOffice, Extragear/graphics, and OSG open source projects. Both KOffice and Extragear/graphics underwent an adaptive maintenance task that involved migration from Qt3 to Qt4. OSG underwent a migration of OpenGL to a new version.

The study uncovered the main result that the commits involved in an adaptive-maintenance task are typically system wide and large. That is, on average adaptive-maintenance tasks touch more files than non-adaptive tasks. For instance, with respect to a file-size measure the obtained result shows that 21% of adaptive commits, compared with 9% of non-adaptive commits, are classified as large commits for KOffice.

In the future, we plan to repeat the study with additional causes of adaptive changes. We also plan to develop a general automated adaptive identification approach via Information Retrieval (IR) methods, such as vector space models (VSM) or latent semantic indexing (LSI). Moreover, we will expand our study by investigating more characteristics regarding adaptive maintenance.

II. REFERENCES

[1] Alali, A., Kagdi, H., and Maletic, J. I., "What's a Typical Commit? A Characterization of Open Source Software Repositories", ICPC, , 2008, pp. 182-191.

[2] Hindle, A., German, D. M., Godfrey, M. W., and Holt, R. C., "Automatic classication of large changes into maintenance categories", ICPC, 2009, pp. 30-39.

[3] Lientz, B. P., Swanson, E. B., and Tompkins, G. E., "Characteristics of application software maintenance", C ACM, vol. 21, 1978, pp. 466-471.

[4] Mockus, A. and Votta, L. G., "Identifying Reasons for Software Changes Using Historic Databases", ICSM, 2000, pp. 120 -130.

[5] Robles, G., Gonzalez-Barahona, J. M., Michlmayr, M., and Amor, J. J., "Mining large software compilations over time: another perspective of software evolution", MSR, 2006, pp. 3-9.