Characterizing Software Architecture Changes: An Initial Study

Byron J. Williams and Jeffrey C. Carver
Department of Computer Science & Engineering
Mississippi State University
{bjw1, carver}@cse.msstate.edu

Abstract

With today's ever increasing demands on software, developers must produce software that can be changed without the risk of degrading the software architecture. Degraded software architecture is problematic because it makes the system more prone to defects and increases the cost of making future changes. The effects of making changes to software can be difficult to measure. One way to address software changes is to characterize their causes and effects. This paper introduces an initial architecture change characterization scheme created to assist developers in measuring the impact of a change on the architecture of the system. It also presents an initial study conducted to gain insight into the validity of the scheme. The results of this study indicated a favorable view of the viability of the scheme by the subjects, and the scheme increased the ability of novice developers to assess and adequately estimate change effort.

1. Introduction

The nature of a software intensive system is that it will change over time. These changes are a crucial aspect of maintenance and have become an ever increasing challenge for software developers. Due in part to the amount of change that occurs, software maintenance has been regarded as the most expensive phase of the software lifecycle. As a system changes, it becomes more complex making it potentially less understandable for the developers and ultimately resulting in decreased system quality.

Late-lifecycle changes can be defined as changes that occur after at least one cycle of the development process has been completed and a working version of the system exists. These unavoidable changes pose an especially high risk for developers. Understanding late-lifecycle changes is important because of their high cost, both in money and effort, especially when they are changes to requirements. Furthermore, these late-lifecycle changes tend to be the most crucial changes because the customers and end-users better understand their needs. Implementation of these changes often causes the system to lose flexibility and deviate from its original design. There are many sources for late-lifecycle changes, including defect repair, changing market conditions, changing software environment, and evolving user requirements. Due to the time pressure caused by these crucial late-lifecycle changes, developers often cannot fully evaluate the impact on the system architecture. As a result, the architecture degrades, leading to lower system quality and making future changes more difficult [2, 5].

When dealing with late-lifecycle changes, it is important to focus on the software architecture, which defines the structure and interactions of the system. When a change affects the system architecture, the original architectural model must be updated to ensure that the system remains flexible and continues to function as originally designed. When a change to the system structure causes the interactions to become increasingly complex, the architecture is likely to degenerate and become un-maintainable. Architectural degeneration leads to a mismatch between the actual functions of the system and its original design. This situation results in confusion for developers which leads to either a major reengineering effort or an early retirement of the system [3].

To address these problems, developers need a way to better understand the effects of a change prior to making the change. This paper proposes a change characterization mechanism that allows developers to conceptualize a change before implementing it will allow them to better predict the effects of the change on the software architecture. The developers can then use that information to come to a consensus on how to implement the change request and take the necessary precautions to prevent architecture degradation. The developers first individually characterize the change

request, then agree on the predicted impact, and finally use historical change data, if available, to compare the impact of similar changes.

The proposed change characterization mechanism builds on research into change classification schemes. The change characterization mechanism is not a classification scheme because it does not attempt to group change requests into separate orthogonal classes. Rather it focuses on helping developers identify specific features of a change that may exhibit certain characteristics.

Classification schemes have been used to assess the impact of changes on source code. Although source code changes often affect the software architecture, there are currently no classification or characterization mechanisms focused specifically on changes to software architecture. Another drawback to classifying changes is that it may be difficult for novice developers to select the correct class or category for a change request. It may also be difficult for experienced developers to reach a consensus on the classification of the specific features of the change request.

This paper presents an initial architecture change characterization mechanism developed to address some of the problems associated with architectural degeneration and the shortcomings of general change classification schemes.. An exploratory study was conducted to assess the usefulness of the scheme and to improve it for further study.

2. Related Work

Change classification schemes have been used by developers to assess the impact and risk associated with making certain types of changes to software. Several benefits of change classification have been identified in the literature, such as using the classification to identify risks associated with change implementation and determining change acceptability. Software change classification schemes also allow engineers to group changes based on different criteria, e.g. the cause of the change, the type of change, the location where the change must take place, and the potential impact of the change. Another benefit of change classification is that it allows engineers to develop a common approach to deal with similar changes, resulting in less overall effort required than if each change was addressed individually [9].

Lientz and Swanson's work identified the frequency of the different types of maintenance activities performed by a large sample of software development organizations [6]. Based on their work and work by Sommerville, the major types of changes identified are: perfective, corrective, and adaptive changes. *Perfective* changes result from new or changed requirements. These changes improve the system to better meet user needs. *Corrective* changes occur in response to defects. *Adaptive* changes occur when moving to a new environment or platform or to accommodate new standards or platforms [10]. Another type of change that often affects system architecture is a preventative change. *Preventative* changes ease future maintenance by restructuring or reengineering the system when a potential problem is identified [7].

The architectural change process described by Nedstam describes the change process as a series of steps [8]:

1. Identify an emergent need
2. Prepare resources to analyze and implement change
3. Make a go/no-go feasibility decision
4. Develop a strategy to handle the change
5. Decide what implementation proposal to use
6. Implement the change.

An architectural change characterization scheme will address steps 3 and 4 by helping developers conceptualize the impact of a proposed change by characterizing the features of the change request.

The change characterization scheme builds on features of existing change classification and analysis schemes that provide insight into changes that affect software architecture. Kung, et al. studied the impact of code changes on the class inheritance structure within a software system [4]. Nedstam, et al. identified changes to be either *architectural*: affecting the structure of the system, *functional*; affecting only user-observable attributes, or *somewhere in between*; affecting both user-observable attributes and the system architecture [8]. In creating the change characterization scheme, all of the above approaches were considered along with others cited in a technical report [11], but only the features that pointed to a direct change to the software architecture were included.

3. Architecture Change Characterization Scheme

The architecture change characterization scheme provides a structured approach to architecture change impact analysis. A developer uses this scheme to characterize a change request starting with high-level characteristics then progressing to a more detailed selection of attributes and their effect on system structure. The high-level characteristics focus on the motivation for the change, the type of change, the size of the change, the impact of the change on static and

dynamic system properties, and finally the functional and non-functional requirements affected by the change. The detailed change characteristics identify the specific architectural changes that must be made to the major architectural views in order to implement the change.

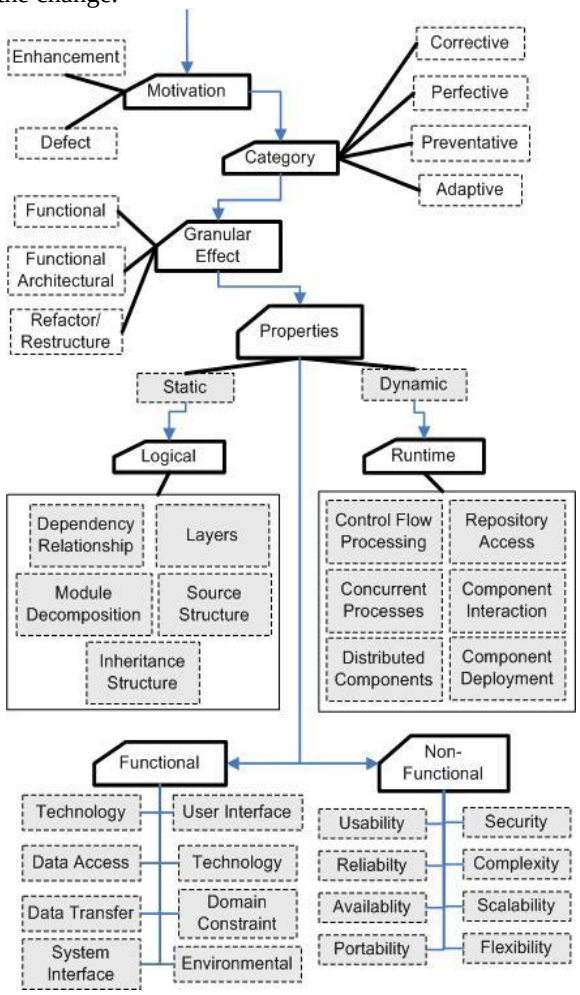

Figure 1: General Change Characteristics

A developer uses an electronic form to record his or her characterization of a change request along with some rationale for the choices. The developer's characterization of the change is then used to facilitate a discussion with other developers about the impact of the proposed change to determine whether the change can be implemented given the existing development constraints and architecture complexity. The scheme has been designed as a decision tree such that choices made for the high-level characteristics affect the potential choices at the more detailed levels. The developer chooses features of the scheme relevant to the architecture being changed. The attributes of the scheme not relevant to the subject architecture are ignored. The initial version of the scheme, as proposed in this paper, will continue to undergo evolution as additional constraints and dependencies among high-level and low-level attributes are discovered.

The architecture change scheme is described in more detail in the following subsections. Section 3.1 and 3.2 describes the general and specific change characteristics, respectively. These sections provide a high-level description of the scheme's attributes. A comprehensive explanation of the creation of the scheme and a detailed description of the attributes and their importance is available in a technical report [11].

3.1 General Characteristics

The first step in characterizing an architecture change is to select the high-level attributes of the change which describe the overall characteristics of the change and its effect on the whole system and development environment Figure 1 shows the general change characteristics. In the figure, the shapes with the bold outline are the general attributes, and the shapes with the dashed line are the values that can be selected for each attribute. The values that are highlighted in gray are measured using the Overall Impact Scale (Table 1). The developer first selects the *motivation* for the change, either an enhancement or a defect. An **enhancement** is a change that improves the system from the point of view of some stakeholder, while a **defect** is a change resulting from an error, fault, or failure.

The next attribute, *category*, determines the type of change. This attribute can be **perfective**, **corrective**, **adaptive,** or **preventative** (described in Section 2).

The *granular effect* of the change explains the depth or size of the change in terms of its architectural impact. There are three options to choose from: **functional**, **functional/architectural**, and **refactor/restructure**. Purely functional changes affect the user-observable attributes and specific system functions. Purely architectural changes (refactor/restructure) are those that affect only the architecture and not a function observable to the user. The functional/architectural change is a change that effects both how the system functions to the user and the architectural structure of the system.

The *properties* attribute determines the effect of the change on the logical and runtime system structures. A **static** change affects the logical system properties, such as the decomposition of modules, module dependency, the system inheritance structure and other system properties that affect the static structure. A change to the **dynamic** properties affects how data is propagated through the system, the behavior of distributed components, how concurrent

processes execute, and other runtime behaviors. For the properties attribute we introduce the Overall Impact Scale (Table 1) to determine the extent of the effect on each property.

Change requests are motivated by a number of different issues. The next list of attributes identify which software engineering issues the change will address in terms of functional and non-functional requirements.

Table 1: Overall Impact Scale

Rating	Name
0	No impact
1	Cosmetic impact
2	Minor impact
3	Substantial impact
4	Major focus of change

The next set of attributes for the general characterization offer more detail into the changes that must be made to the system architecture. The developer can choose the logical and runtime architectural views that must be changed in order to implement the change request.

The *logical* attribute includes a comprehensive list of general architecture characteristics that can be used to describe the static framework of most object oriented software intensive systems. The list of logical architecture characteristics includes; **dependency relationships**, **layers**, **inheritance structure**, **module decomposition**, and **source structure**. The exact changes made to each view of the architecture will be described in more detail during the specific characterization. At this point, the developer identifies the overall impact, if any, to each architecture characteristic.

The *runtime* attribute serves a similar purpose as the logical attribute. This attribute lists the dynamic architecture characteristics common to most object oriented software intensive architectures. The list of general runtime views include; **control flow processing**, **repository access**, **concurrent processes**, **component interaction**, **distributed components**, and **component deployment**.

The *functional* issues include **technology**, **data access**, **data transfer**, **system interface**, **environmental**, **user interface**, **and domain constraints**. The *non-functional* issues are common areas where the goal of the software change is to improve on some quality attribute. These issues include **usability**, **reliability**, **availability**, **security**, **portability**, **complexity**, **flexibility**, and **scalability**.

The general architecture change characterization aims to provide the developer with a means of describing a change request in terms of its overall affect on the system. The specific characterization which provides more detail into the changes required to the logical and runtimes structures is described in the next section.

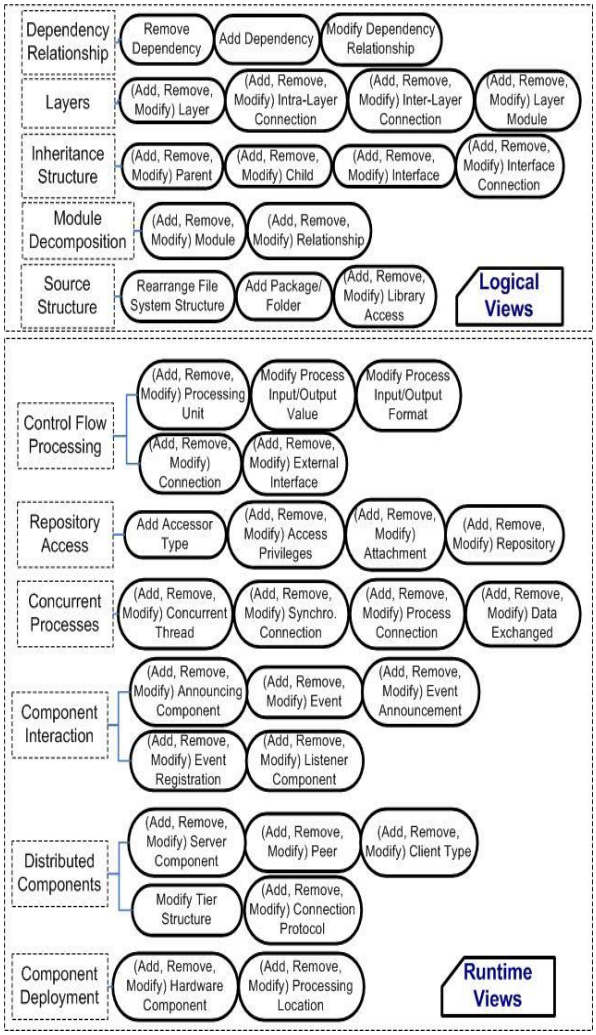

Figure 2: Specific Change Characteristics

3.2 Specific Characterization

The specific architecture change characterization allows the developer to analyze the architecture while making recommendations for changes to the overall structure in order to implement the change request. The changes that are reflected in the architecture include changes to any architecture module, interface, component, or connections between modules and components. Each rating will correspond to the type of change applied to an item in the logical and runtime lists of architecture characteristics. Figure 2 provides a visual overview of the specific change characteristics

and the types of changes that can be made to elements in each architecture view. These changes include adding, modifying, and removing elements and/or the connections between the elements.

The Specific Impact Scale found in Table 2 describes the magnitude of the changes that can be made to each of the architecture structures listed. For each type of change, the developer selects a value from the Specific Impact Scale that identifies the magnitude of the change.

Table 2: Specific Impact Scale

Rating	Name
0	No impact
1	Small impact–single mod./comp.
2	Small impact – multiple mod./comp.
3	Significant impact–single mod./comp.
4	Significant impact–multiple mod./comp.

4. Study Description

The main goal of this study was to gain insight into the feasibility and usefulness of the architecture change characterization scheme. Stated formally, in GQM format, the goal was:

Analyze the architecture change characterization scheme in order to understand it with respect to usability, viability, and architecture impact estimation from the point of view of the researcher in the context of a classroom study

The study focused on architectural impacts because architectural changes tend to have an adverse effect on system quality when performed without taking the necessary precautions to prevent degradation. The questions addressed include:

1. How well did the change characterization by the subjects match the change characterization by the researchers?
2. Is the characterization scheme easy to use?
3. Do changes that exhibit different characterizations require different amounts of effort to implement?
4. Does the scheme support effort estimation?
5. Does the scheme add value to the change process?
6. Does the scheme facilitate communication amongst developers?

Answers to these questions will provide insight into whether the subjects understand the scheme and help to identify its strengths and weaknesses.

4.1 Study Setup

The study was conducted in the Software Architecture and Design Paradigms class at Mississippi State University. This class focuses on software architecture development methodologies and analysis methods including model representations, component-based design, design patterns, and frameworks. There were 25 subjects (22 seniors and 3 graduate students) who participated in the study.

During the experimental tasks, described in Section **Error! Reference source not found.**, the subjects worked with the artifacts from the Tactical Separation Assisted Flight Environment (TSAFE), a tool designed to aid air-traffic controllers in detecting and resolving short-term conflicts between aircraft. The TSAFE source code contains 80 java classes and 20k lines of source code. Prior to the beginning of the study, each subject had already created their own version of an architecture document using the TSAFE requirements. The process of creating their own TSAFE architectures helped them become familiar with the system.

4.2 Training and Experimental Tasks

Before using the characterization scheme, the subjects were given three 1-hour training sessions. The first session provided a general overview of software changes and highlighted the importance of designing flexible architectures that could readily handle change. In the second session, the subjects were given the requirements and architecture for a sample system. They were shown sample change requests and then modified the architecture to address the requests. This session gave the subjects hands-on experience making architecture changes. The third and final training session focused on explaining the purpose for the characterization scheme, defining the attributes, detailing its use, and allowing the subjects to use it on several examples. This session ended with a discussion of the characterization scheme to answer any questions that may have arisen during the training session.

The change classification study took place during the final two homework assignments of the semester. The subjects were first given feedback on the TSAFE architectures that they created earlier in the semester, and then were given the "gold standard" TSAFE architecture created by the authors to be used for the study. To make the assignments tractable, the subjects were only required to change the architectural diagrams and not change the actual source code. The actual implementation of each source code change was done by one of the authors. The resulting architecture and code was used as a basis of comparison with the subject's changes during analysis.

After the second training, the subjects were given a single change request to complete for the first

homework assignment. For this task, the subjects were required to analyze the architecture, change the architecture diagrams, record the details of the change, and provide justification and rationale. After each subject completed their individual changes, they were randomly assigned a partner. Each pair repeated a similar process as was done individually. The subjects turned in their updated group architecture diagrams and detail forms along with a report describing their interaction and comparing the architecture created by the pair to the ones created individually.

Table 3: Training and Experimental Tasks

Task	Description	Time
T1	Software change overview	1-hr
T2 T2.1	Architecture change exercise Review "gold standard" arch.	1-hr
A1 A1.1 A1.2 A1.3 A1.4	Individual arch. change Record change detail Group arch. change Record change detail Submit experience report	1-wk
T3 T3.1	Change characterization training Characterization exercises	1-hr
A2 A2.1 A2.2 A2.3 A2.4 A2.5 A2.6	Individual arch. change Characterize changes Record change detail Group arch. change Characterize changes Record change detail Submit experience report	2-wk
A3	Post-study survey	1-hr

Table 4: Study Change Requests

# - Name	Description & Impact
1 – Confor-mance Monitor	Calculate whether flights are on set courses and visually alert ATC if not. Add module, determine interface, and change GUI classes.
2 – Feed Display	Add connections to data feed to display raw flight coordinates to ATC. Transfer data from low-level classes that handle raw flight data to GUI modules.
3 – Loss of Separation Detector	Visually alert ATC when 2 flights are within certain distance from each other. Add module, determine interface, and change GUI classes.

In the second homework, the subjects were given two change requests to make on the TSAFE architecture. To ensure as much consistency among subjects as possible, the subjects were always asked to return to the original "gold standard" version of the architecture before making any changes (i.e. the changes did not build on each other). This assignment was given after the third training session, allowing the subjects to use the change characterization process. The subjects performed the same steps as in the first assignment plus the additional step of characterizing the change requests with the change characterization scheme. The steps followed by the subjects included: characterization of the change request, modification of the architecture diagrams, and documentation of the change. The subjects were then assigned a different partner to perform the changes as a group. Again, each pair had to come to a consensus on the change characterization, implementation detail, and provide a description of their experiences. They were asked to describe how they used the characterization scheme and any differences between their individual characterization and changes and the group ones.

At the end of both tasks, the subjects were given a survey on their opinions about the difficulty of each change, the ease of using the change characterization scheme and whether the change scheme was beneficial to the process. Table 3 lists the study tasks which includes trainings and homework activities. Table 4 lists the change requests used in the study.

4.3 Data Collection

To address the questions of interest (Section 4), both qualitative and quantitative data was collected. The qualitative data was obtained from questionnaires, surveys, and experience reports submitted by the subjects. The quantitative data was provided by the subjects through electronic forms recording the implementation details for each change, including the number of modules and components changed and which architecture views would be affected by the change. Finally, the modified architectures were collected from each subject to analyze the exact changes made to the system architecture.

5. Study Results and Analysis

This section is organized around the six research questions posed in Section 4. For each question, the relevant qualitative and quantitative data is presented.

1. How well did the change characterization by the subjects match the change characterization by the researchers?

The two changes from Assignment 2 (Change 2 and 3) were characterized by one of the authors prior to the study (the "gold standard"). These values are shown in Figure 3 and Figure 4. The answer for this question comes from the characterization data

submitted by the subjects for Changes 2 and 3. If the subjects correctly understood the attributes of the characterization scheme and how the changes requests would affect those attributes, then their characterization of the changes should have been similar to the "gold standard" characterization. Any discrepancies in the result are likely caused by a partial or complete misunderstanding of the attributes of the characterization scheme and/or the TSAFE architecture that was provided to the subjects.

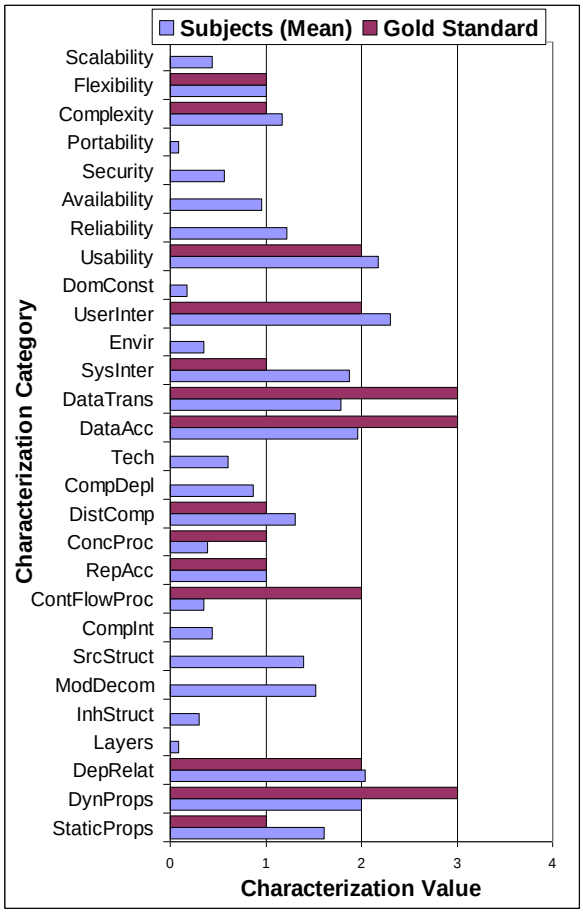

Figure 3: Characterization Comparison - Change 2

In order to determine the "closeness" of the characterizations, the mean of values of the subject's characterizations were computed. This value was compared to the "gold standard" for each change. The subjects characterizations were viewed to be "close" to the "gold standard" if the (absolute value) of the difference between the two values was less than or equal to 1. Of the 28 general characteristics, 22 attributes met the standard for Change 1 and Change 2. Figure 3 and Figure 4 show a comparison of mean subject values to the "gold standard" for Change 2 and Change 3 respectively. Based on these results, a majority of subjects seemed to understand how to use the scheme.

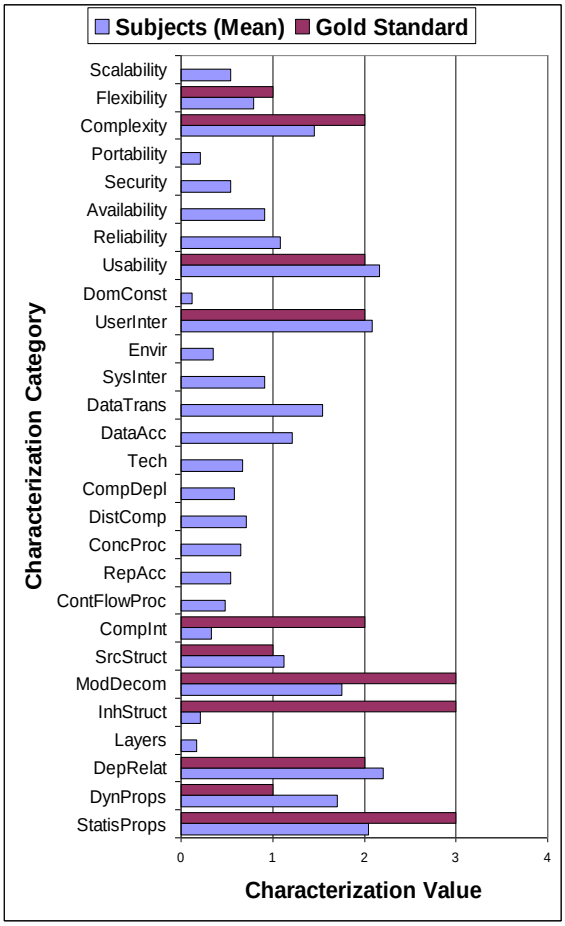

Figure 4: Characterization Comparison - Change 3

2. Is the characterization scheme easy to use?

In the survey completed after the two assignments, the subjects were asked to indicate their level of agreement with the following statements about the usefulness of the classification scheme:

1. The attributes are logical and easily understood
2. The scheme is beneficial for a developer making a change
3. I understood the effect of the changes to the system architecture better using the scheme than without it
4. The scheme was detailed and covered all aspects of the architectural implementation
5. The change scheme helped me to understand the impact of the change request

For each question, a 5-point Likert rating scale was used, ranging from 1 - totally disagree to 5 - totally agree. Figure 5 shows these results.

The generally positive results provide support for the idea that characterization scheme is both useful

and practical. The results were analyzed the results using a one-sample t-test with a test value of 3 representing a neutral response. The results of the test are show in Table 5. The degree of freedom for each test is 24. The results show a mean response greater than 3 and this value is statistically significant

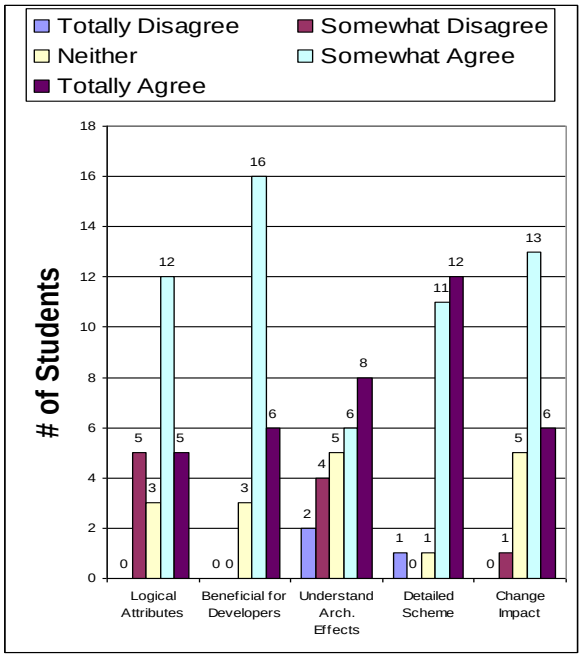

Figure 5: Subject's Survey Results

Table 5: Statistical Survey Results

Statement	Mean	T-Value	P-Value
1.	3.68	3.302	.003
2.	4.12	9.333	.000
3.	3.56	2.133	.045
4.	4.32	7.333	.000
5.	3.96	6.080	.000

3. Do changes that exhibit different characteristics require different amounts of effort to implement?

Based on the "gold standard" characterization shown in Figure 3 and Figure 4, Change 2 and 3 were characterized differently.

Change 2 was a *perfective enhancement* that was a *functional* changed that did not require the addition of any modules. All of the information required to display the feed data was already being processed by the system. The change request only required some portion of the data (the feed) to be propagated through the system to reach the GUI display function. This change required the modification of a small number of LOC in a relatively large number of modules.

Change 3 was also a *perfective enhancement* that was both a *functional* and *architectural* change because it required the addition of a significant architecture module. To use of this module, a parent module and other modules that need to interact with the new module had to be changed to provide the additional functionality. Several lines of code were also added to a user interface module so that the results of calculations could be displayed graphically to the user when the function is triggered. We used a tool to compare the source code differences of each changed TSAFE implementation to the original in order to determine the number of modules changed and the number of LOC changed for each change. Table 6 shows the change implementation detail for Change 2 and Change 3. Changes 2 and 3 had different characterizations. This difference resulted in a differing amount of implementation effort.

Table 6: Change Detail

Detail	Change 2 Feed Display	Change 3 LOS Detector
Modules Modified	7	7
Modules Added	0	1
LOC Mod./Add.	37	190

4. Does the scheme support effort estimation?

For each change the subjects were required to estimate the number of module and component changes that would be required. First, the subjects were asked which change would require the most effort to implement. A majority of 16 subjects identified Change 3 as the most difficult, 6 subjects Change 2, and the remaining 3 chose Change 1. Changes 1 and 3 were very similar changes in terms of their impact to the architecture and actual implementation detail (both were implemented by adding one module and modifying 7). We hypothesize that the majority of the subjects chose Change 3 as the most difficult because of the rigor of the change characterization process in forcing them to consider which aspects of the architecture would be affected.

When comparing the results for Changes 1 and 3 (which were similar in terms of the actual implementation), the subjects estimated that a different number of modules would have to change. The mean number of module changes estimated for Change 1 was 1.84 and for Change 2 was 2.88. This difference was statistically significant (t_4 = -2.153, p=.036 [t-test]; Z = -2.399, p=.016 [Mann-Whitney]). This result suggests that the subjects were able to identify additional architectural changes when using the

characterization scheme (Change 3) that were not apparent without the change scheme (Change 1).

5. Does the scheme add value to the change process?

In the post study survey the subjects provided their opinions of the scheme, how the scheme could be improved, and any problems that they encountered while using the scheme. These questions were used to elicit information to help with improvement of the characterization scheme and to better understand its strengths. In the list below the number of subjects who gave each response is in parenthesis. The subjects said that the characterization scheme:

- Aids in determining what changes should be made to each architecture view and the impact the change will have on the view (7);
- Helped ensure thoroughness of change detail (6);
- Would be a good communication tool for project managers, software architects, maintainers, and developers (6);
- Is good for large changes but not practical for small changes (5);
- Has too many attributes (5);
- Requires more training in its use than was provided (3);
- Is complete with the right level of detail (2);

6. Does the scheme help facilitate communication amongst developers?

For both assignments, the subjects first worked individually then with a partner. These two steps were used to capture the interaction between the subjects to determine if the characterization scheme facilitated the discussion of the impact of a change request.

The reports were analyzed and coded to extract information about the use of the characterization scheme during the group meeting. Each group did not specifically comment on their use of the characterization scheme in the group meeting, but any statements about to the use of the scheme were extracted from the experience report. Some of the comments (paraphrased) made in the reports include:

- Four groups reported that they recorded the characterization of the changes after discussing their individual change rationale. Next, they determined how their individual changes compared to the changes made jointly using the scheme. Finally, they recorded the change detail reflected by the scheme and updated the architecture diagrams to reflect this new combined architecture. They used the scheme to determine the change detail.
- One group used the characterization scheme as a checklist while recording the architecture changes on to the change detail form. The group stated that the scheme helped their decision process by focusing their discussion on which changes listed for each view were needed.
- Three groups did not use the scheme to make the actual decisions. They simply used it at the end of the process to record the characterization of the changes after their change decisions were made.
- Two groups used the characterization scheme after their analysis but prior to modifying the architecture diagrams. By using the characterization scheme at this point, they were able to determine what changes they would have to make to each architecture view.

6. Study Implications

The purpose of this initial study was to assess the viability of an architecture change characterization scheme designed to assist developers in estimating the potential effect of a change on an architecture. In Section 5, we presented qualitative and quantitative data to address a set of research questions. The main contribution of this study is the presentation and analysis of an initial architecture change characterization scheme. The subjects found the scheme to be useful and commented that it has practical application in a development environment. The characterization scheme also increased the ability of novice developers to analyze the complexity and difficulty of a software architecture change.

We also wanted to determine whether changes with different characterizations would require different amounts of effort to implement. This result motivates the use of the change characterization scheme as an input for effort estimation. Predicting software change effort is a difficult task even for experienced developers. Therefore, the purpose for creating this scheme was to provide input to a decision support model that will incorporate change characterization, impact analysis, and risk assessment to aid developers in making go/no-go decisions for changes based on how the system will be affected.

The results obtained did, in general, support the conclusion that differently characterized changes require different amount of effort. That is, Change 3 required significantly more modules and LOC to be changed than Change 2. The subjects also qualitatively agreed that Change 3 was more difficult.

7. Threats to validity

This section discusses the threats to validity that were present both in the design and in the execution of the study and their potential impact on the results.

Using Students to Perform Analysis: Students are frequently used in empirical studies to provide some evidence of the usefulness of software engineering products and processes [1]. Students were able to provide data about the use of the scheme and answer some important questions about it. The threats associated with using students in this study include their potential bias in answering survey questions for fear of criticizing. They also may not have the appropriate experience to evaluate the scheme's usefulness in a professional setting.

Conclusions about Scheme Estimation Support: The results in Section 5 indicated that the use of the characterization scheme helped the subjects correctly identifying the change that would require the most implementation effort. There was also a significant difference in the number of modules changes estimated by the subjects when using the scheme than when not using it. This difference could have been caused by a learning effect. When performing the third change, the subjects were more familiar with the process and architecture and better able to asses the change.

Change Differences from Other Factors: The different amounts of effort required to implement Changes 2 and 3 could have been caused by factors other than the difference in the change characterization. Because the subjects only characterized the 2 changes (and did not implement them), we were unable to investigate other possible differences that could have caused the different amounts of effort.

8. Summary and Future Work

We will continue to refine the characterization scheme by making changes based on the study feedback. We will characterize changes in historical datasets that include implementation detail that can be used for validation. This activity will allow us to identify trends about change characteristics in a particular system, and recommend best-practices for future changes with similar characteristics.

Change characterization can be a useful tool in determining the impact on the system. After further research, we envision that this characterization mechanism could be incorporated in to an organization's change implementation process. An additional step could be added after receiving a change request to allow the developers to characterize that change request.

Being able to accurately identify changes that will affect software architecture will aid developers in understanding the change impact and help them make architecture changes without degrading the quality of the system.

9. Acknowledgements

We thank the students in Software Architecture and Design Paradigms who participated in this study. This work is supported by NSF Grant CCF-0438923.

10. References

[1]. J. Carver, L. Jaccheri, S. Morasca, and F. Shull. "Issues in Using Students in Empirical Studies in Software Engineering Education," in *Proceedings of the Ninth International Software Metrics Symposium*, 2003, pp. 239-249.

[2]. S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does Code Decay? Assessing the Evidence from Change Management Data," *IEEE Transactions on Software Engineering*, v. 27, no. 1, 2001, pp. 1-12.

[3]. L. Hochstein and M. Lindvall, "Combating Architectural Degeneration: A Survey," *Information and Software Technology*, v. 47, no. 10, 2005, pp. 643-656.

[4]. D. Kung, J. Gao, P. Hsia, F. Wen, Y. Toyoshima, and C. Chen. "Change Impact Identification in Object Oriented Software Maintenance," in *Proceedings of the International Conference onSoftware Maintenance*, Victoria, BC, 1994, pp. 202-211.

[5]. M. M. Lehman and L. Belady, *Software Evolution - Processes of Software Change*. London: Academic Press, 1985

[6]. B. Lientz and B. Swanson, *Software Maintenance Management* Addison-Wesley, 1980

[7]. P. Mohagheghi and R. Conradi. "An Empirical Study of Software Change: Origin, Acceptance Rate, and Functionality Vs. Quality Attributes," in *Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE '04)*, 2004, pp. 7-16.

[8]. J. Nedstam, E. A. Karlsson, and M. Host. "The Architectural Change Process," in *Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE '04)*, 2004, pp. 27-36.

[9]. N. Nurmuliani, D. Zowghi, and S. P. Williams. "Using Card Sorting Technique to Classify Requirements Change," in *Proceedings of the 12th IEEE International Requirements Engineering Conference*, 2004, pp. 240-248.

[10]. I. Sommerville, *Software Engineering*. 7th ed: Addison-Wesley, 2004

[11]. B. Williams and J. Carver, "Characterizing Changes to Assess Architectural Impact", in *Technical Report MSU-070115*, Department of Computer Science and Engineering, Mississippi State University, 2006.