UNDERSTANDING THE CAUSES OF ARCHITECTURE CHANGES USING OSS MAILING LISTS*

Wei Ding1,2† and Peng Liang1‡
1State Key Lab of Software Engineering, School of Computer Science
Wuhan University, 430072 Wuhan, China
2Key Laboratory of Earthquake Geodesy, Institute of Seismology
China Earthquake Administration, 430071 Wuhan, China
lingwhere@whu.edu.cn
liangp@whu.edu.cn
Antony Tang3
3Faculty of Science, Engineering and Technology
Swinburne University of Technology, VIC 3122 Melbourne, Australia
atang@swin.edu.au
Hans van Vliet4
4Department of Computer Science
VU University Amsterdam, 1081 HV Amsterdam, The Netherlands
hans@cs.vu.nl

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

The causes of architecture changes can tell about why architecture changes, and this knowledge can be captured to prevent architecture knowledge vaporization and architecture degeneration. But the causes are not always known, especially in open source software (OSS) development. This makes it very hard to understand the underlying reasons for the architecture changes and design appropriate modifications. Architecture information is communicated in development mailing lists of OSS projects. To explore the possibility of identifying and understanding the causes of architecture changes, we conducted an empirical study to analyze architecture information (i.e., architectural threads) communicated in the development mailing lists of two popular OSS projects: Hibernate and ArgoUML, verified architecture changes with source code, and identified the causes of architecture changes from the communicated architecture information. The main findings of this study are: (1) architecture information communicated in OSS mailing lists does lead to architecture changes in code; (2) the major cause for architecture changes in both Hibernate and ArgoUML is preventative changes, and the causes of architecture changes are further classified to functional requirement, external quality requirement, and internal quality requirement using the coding techniques of grounded theory; (3) more than 45% of architecture changes in both projects happened before the first stable version was released.

* This paper is an enhanced version of our previous paper “Causes of Architecture Changes: An Empirical Study through the Communication in OSS Mailing Lists” published at the SEKE 2015 conference [1].
‡ Corresponding author.
1. Introduction

Software architecture (SA) represents “the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution” [2]. Systems continuously evolve and change to be adapted to new uses, just as buildings change over time [3], which consequently leads to architecture changes. Understanding the causes of architecture changes is important to help practitioners to understand the knowledge of the design decisions that lead to the architecture changes [4], and also the knowledge about the causes of architecture changes can be captured to prevent architecture knowledge vaporization and architecture degeneration [5]. The causes of architecture changes are regarded as an essential element of architectural design decision, which is a first-class entity to represent architecture [6], and are used to develop related methods to deal with specific architecture changes, for example, architects analyze due to what cause the property of an architecture is inhibited in order to transform the architecture to satisfy non-functional requirements [7]; architectural styles as analysis tools are used to analyze the causes of architecture changes, and in turn to predict the effect of the architecture changes [8]. Architectural knowledge vaporization (e.g., design decisions and causes of architecture changes) will lead to increased maintenance costs [6]. To prevent this problem, developers (especially architects) need a way to record and communicate the causes of changes in architecture. With an explicit description of architecture as well as their changes [9], software maintainers can better understand the ramifications of architecture changes. An improved understanding can help software maintainers more accurately analyze the impact and estimate costs of modifications [10]. But the reality is that the rationale of architectural design decisions (e.g., their causes) is often not available in SA documentation [11], especially in OSS development when architecture design is rarely documented (only 5.4% of 2000 investigated OSS projects have some SA documentation) [12]. Causes of architecture changes are communicated between developers through intermediary media such as mailing list, especially in a distributed development context when face-to-face communication is difficult. Mailing list is an important social media for knowledge sharing between knowledge providers and knowledge seekers in OSS projects [13]. Our recent study has shown that communication on architecture does exist in the mailing lists of two popular OSS projects (Hibernate and ArgoUML) [14], and OSS development mailing lists may act as a potential source to extract and identify the cause information of architecture changes in a project.

One of the characteristics of many successful OSS projects is the existence of a SA [15]. Architecture change is also a widespread phenomenon in OSS development, for example, an investigation of the changes in Linux kernel’s evolution indicates that most remarkable growth for a “stable” version has been in the addition of new features and
support for new architectures rather than fixing defects [16]. To understand the causes of architecture changes [17][18], we conducted an empirical study to extract, identify, and analyze the architecture change information communicated in the OSS mailing lists of two popular OSS projects: Hibernate and ArgoUML based on the data (i.e., architectural threads, which are a set of communication posts on the same topic that contain architecture information in mailing lists) we collected in [14]. The identified architecture changes in mailing lists were further located and verified (confirmed) in the source code of the two projects, and the causes of the architecture changes were classified through the communicated content in architectural threads. The goal of this work is to provide a practical understanding of the causes of architecture changes through communication in OSS mailing lists: Does architecture communication in mailing lists lead to architecture changes in source code? What types of causes that lead to the architecture changes? When do OSS developers communicate the causes of architecture changes? Note that, identifying architecture changes through OSS mailing lists is only one way, and uses only one source, to identify architecture changes in an OSS project.

To answer these questions, we first extracted architecture change information from the architectural threads of OSS mailing lists and further classified the causes of architecture changes with a top-down approach (i.e., using an existing categorization of causes of architecture changes provided in [17]), then checked and verified these changes against source code. We conducted this study based on the architectural threads collected in two popular OSS projects: Hibernate and ArgoUML [14]. These architectural threads in Hibernate and ArgoUML are used to extract, identify, and analyze the causes of architecture changes. The results show that the major cause for architecture changes in both Hibernate and ArgoUML is preventative changes, which ease future maintenance by restructuring or reengineering the system.

A further analysis of architecture change cause based on the collected architectural posts using the coding techniques of grounded theory, and a detailed description and explanation of the study process and data analysis in the methodology part, are the major extensions to our previous work presented in [1].

The rest of this paper is organized as follows. A brief review of related work is discussed in Section 2. The methodology, including research questions and study process, is described in Section 3. The results of this study are presented and discussed in Section 4. Threats to validity are discussed in Section 5. We conclude and outline the future directions of this work in Section 6.

2. Related Work

2.1. Cause of architecture change

The causes of architecture changes have been explored in software development in various perspectives. The work in [17] uses a systematic literature review to characterize architecture changes from existing literatures. As part of the Software Architecture
Change Characterization Scheme (SACCS), a general classification of causes of architecture changes presented in [17] and another work [18] by the same authors can be used as the basic categorization of the causes of architecture changes in the two OSS projects in this study, which is elaborated in Section 3. The work in [19] analyzes group interviews in various workshops for different levels of participants, e.g., developers, testers, and architects in five companies. The results validated a taxonomy of the causes for architecture technical debt, a kind of architecture inconsistency, which can be incurred and repaid by architecture changes. The work in [20] uses various versions of an ATM simulator to observe and analyze what happens when a system evolves and new requirements are added. The results of this work show that changes in requirements may lead to architecture changes and drift, and consequently developers (architects) that do not fully understand the design may take sub-optimal decisions, resulting in design erosion. The authors also identified the causes of design erosion, which can be the causes for architecture changes as well. The work in [21] developed an approach ARCADE (Architecture Recovery, Change, And Decay Evaluator) to analyze architecture changes existing in hundred versions of 14 open source Apache systems. The results show that a system’s versioning scheme is not an accurate indicator of architecture change and important architecture changes are hidden in corresponding structure-based views. However, it is not easy to analyze and extract the cause information of architecture changes using the approach in [21].

Our work in [12] analyzed 108 OSS projects that have architecture documentation in order to understand what SA information has been documented. The result shows that only 16.7% of OSS projects (18 out of 108) have architecture rationale information, which contains partially useful cause information to explain architecture changes. Considering this situation, we try to identify the cause information of architecture changes from other media, i.e., development mailing lists, in this work.

### 2.2. Communication through mailing lists in OSS projects

Mailing lists in OSS development, as a rich source of communication of development, have been investigated in many studies. The work in [13] discusses the altruistic sharing of knowledge between knowledge providers and knowledge seekers in the developer and user mailing lists of Debian project. The authors developed the Knowledge Sharing Model (KSM) to show how knowledge can be shared (communicated) in OSS mailing lists, and used email exchanges between mailing list participants as quantifiable measures of knowledge sharing activities in OSS development. Some keywords in the subjects of posts of mailing lists are used to identify posting and replying posts, e.g., “Re:”. The study in [22] examined the first posts of newcomers in the mailing lists of four popular OSS projects: MediaWiki, GIMP, PostgreSQL, and Subversion. The authors found that knowledge communication (nearly 80% of newbie posts received replies) was positively correlated with their future participation. Mockus and his colleagues used email archives of source code change history and problem reports to quantify aspects of developer participation, core team size, code ownership, productivity, defect density, and problem
resolution intervals, for two large OSS projects, Apache and Mozilla [23]. These works pay attention to all the posts and threads in a mailing list during a certain period, while our study specifically extracts, identifies, and analyzes architecture changes and their causes through the communication in mailing lists. As mentioned in Section 1, our previous work extracted architectural threads collected in the mailing lists of two popular OSS projects: Hibernate and ArgoUML [14]. The difference between [14] and this work is that the work in [14] covers all architecture information communicated through mailing lists, while this work focuses on the cause information of architecture changes in mailing lists.

3. Methodology

To explore the causes of architecture changes through the communication in OSS mailing lists, we select and analyze the mailing lists of two popular OSS projects: Hibernate and ArgoUML, based on the data (i.e., architectural threads) collected in our recent work [14]. In this section, we describe the design of this study with following components: the objective and research questions are presented in Section 3.1, the selection criteria of the OSS projects are described in Section 3.2, and the study process is elaborated in Section 3.3. Data analysis methods are presented in Section 3.4.

3.1. Objective and research questions

The goal of this study, formulated using the Goal-Question-Metric (GQM) approach [24] is: to analyze architecture changes through the communication in mailing lists for the purpose of characterizing the causes of architecture changes from the point of view of OSS developers in the context of OSS development. We formulate the following research questions (RQs) based on the abovementioned goal.

RQ1: What are the causes that lead to architecture changes in OSS development?

**Rationale:** Mailing lists have been used as a major vehicle for the communication in OSS development [25]. Architecture information is communicated in the mailing lists of OSS projects [14]. Some of them may discuss specific architecture information, e.g., the causes of architecture changes. With the existing categorization of architecture changes provided in [17], we want to understand in a practical perspective the causes of architecture changes in OSS development through the communicated content extracted from architectural threads. Knowledge and understanding about the causes of architecture changes (evolution) as well as their risks can facilitate the development of strategies to mitigate these risks in software evolution [26].

RQ2: What are the trends of causes that lead to architecture changes in a time perspective?

**Rationale:** We intend to identify when various types of architecture changes happened and their causes were communicated in a time perspective. The answer of this question would allow us to further identify the best timing for performing treatments to
various types of causes of architecture changes, and help practitioners to understand the distribution of various causes of the changes in the development lifecycle. To investigate the relationship between the causes of architecture changes and time of releases, the studied period of both OSS projects is divided into two stages according to their first stable releases, i.e., ArgoUML v0.10\(^d\) and Hibernate v1.0 final\(^b\). The first stable version of an OSS project is typically when the OSS is mature enough to be released. We want to understand how software architecture changes and what differences exist on these changes before and after the first stable version is released.

### 3.2. Selection criteria of OSS projects

Three criteria are used in this study to select OSS projects that have mailing lists: (1) The duration of the project is more than 10 years. (2) There are more than 1000 posts in the development mailing list of the project, which provides rich data to mine architecture information. (3) There are more than 50 developers who ever used the mailing list, which is meaningful to analyze the behavior of the developers on communicating architecture information using the mailing list.

Based on these selection criteria, we chose Hibernate and ArgoUML as the OSS projects which mailing lists were analyzed. Hibernate provides an Object/Relational mapping (ORM) framework which implements the Java Persistence API, and is popularly used in Java applications. Hibernate has 20,413 posts in its development mailing list between Jan 2002 and Aug 2014, when the release version was evolved from v0.8.1 to v4.3.6. Note that the mailing list of Hibernate was migrated from Sourceforge to JBoss in Aug 2006. The mailing list of Hibernate in Sourceforge was initially maintained by a core developer, but he did not continue this administration effort after 11-Aug-2006\(^c\). Hence, the investigated time period of the mailing list of Hibernate in this study covers the time period of two mailing lists hosted at Sourceforge and JBoss respectively. ArgoUML is a popular open source UML modeling tool developed in Java. ArgoUML accumulates 26,439 posts in its development mailing list between Jan 2001 and Aug 2014, when the release version was updated from v0.8 to v0.34\(^d\). The information of the Hibernate and ArgoUML development mailing lists are elaborated in Table 1.

<table>
<thead>
<tr>
<th>Project</th>
<th>Mailing List URL</th>
<th>Time Period</th>
<th>Num. of Posts</th>
<th>Num. of Arch. Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hibernate</td>
<td><a href="http://sourceforge.net/p/hibernate/mailman/hibernate-devel/">URL</a></td>
<td>Jan 2002 - July 2006</td>
<td>8,913</td>
<td>80</td>
</tr>
</tbody>
</table>

\(^a\) [URL](http://argouml.tigris.org/servlets/NewsItemView?newsItemID=128)
\(^b\) [URL](http://sourceforge.net/p/hibernate/mailman/message/5497028/)
\(^c\) [URL](http://sourceforge.net/p/hibernate/mailman/message/13328372/)
\(^d\) v0.34 is a released version after v0.8 in the version history of ArgoUML. The core developers of ArgoUML originally planned to release ArgoUML from version v0.8 to v0.9, then to v1.0. However, they decided to use version number v0.10 instead of v1.0 in order to encourage developers to continuously contribute to the project.
Understanding the Causes of Architecture Changes using OSS Mailing Lists

<table>
<thead>
<tr>
<th>Project</th>
<th>Mailing List URL</th>
<th>Time Period</th>
<th>Num. of Posts</th>
<th>Num. of Arch. Threads</th>
</tr>
</thead>
</table>

The IEEE 1471-2000 standard suggests ten main architecture elements for architectural description [27], which were employed as the categorization of architecture elements in our prior work to identify various architecture elements documented in an architecture document [12]. In this study, we also use this categorization to identify various architecture information communicated in mailing lists, i.e., record the types and contents of architecture elements in architectural posts. Any post that contains architecture elements is identified as an architectural post and one architectural thread is composed of one or several architectural posts with the same topic. The description of the ten architecture elements defined in IEEE 1471-2000 is presented in Table 2.

Table 2. Architecture elements in IEEE 1471-2000 standard [27]

<table>
<thead>
<tr>
<th>Architecture Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>A collection of components organized to accomplish a specific function or set of functions.</td>
</tr>
<tr>
<td>Mission</td>
<td>A use or operation for which a system is intended by one or more stakeholders to meet some set of objectives.</td>
</tr>
<tr>
<td>Environment</td>
<td>Determines the setting and circumstances of developmental, operational, political, and other influences upon that system.</td>
</tr>
<tr>
<td>Stakeholder</td>
<td>An individual, team, or organization that has interests in a system.</td>
</tr>
<tr>
<td>Concern</td>
<td>Interest which pertains to the system’s development and is related to one or more stakeholders.</td>
</tr>
<tr>
<td>Model</td>
<td>Developed using the methods established by its associated architectural viewpoint.</td>
</tr>
<tr>
<td>Rationale</td>
<td>Records explanation, justification, or reasoning about architectural decisions that have been made in architecture design.</td>
</tr>
<tr>
<td>View</td>
<td>A representation of a whole system from the perspective of a related set of concerns.</td>
</tr>
<tr>
<td>Viewpoint</td>
<td>A specification of the conventions for constructing and using a view.</td>
</tr>
<tr>
<td>Library viewpoint</td>
<td>A viewpoint that is defined in other literature and can be directly used (referred), e.g., the 4+1 view model.</td>
</tr>
</tbody>
</table>

3.3. Study process

This study was conducted in three phases. The first phase was data collection. We first identified the architectural threads in the mailing lists from the two OSS projects selected in Section 3.2 (the numbers of architectural threads identified in Hibernate and
ArGoUML respectively are shown in Table 1). Thirty architectural threads were initially selected as the data for a pilot study. In this pilot study, we executed Step1 to Step3 as described below to mitigate personal bias on understanding and identifying architecture changes and their causes. Following the pilot study, we again executed Step1 to Step3, to extract, classify, and review the causes of architecture changes in the remaining architectural threads in the mailing lists of Hibernate and ArGoUML.

**Step1**: manually identifying architectural threads that lead to architecture changes by checking the content of posts in mailing lists;

**Step2**: Extract architecture changes and further classify the causes of architecture changes with a top-down approach (i.e., following an existing categorization in [17]);

**Step3**: Review the identified and classified types of causes of architecture changes by two researchers to partially mitigate the threat of personal bias;

The second phase involved the verification of architecture changes in source code. The Classes, Packages, or architecture design discussed in mailing lists are checked and located in source code. A semi-automatic static code analysis tool Understand was used to identify the changes of source code. The Understand tool can identify the existence of a specific Class or Package in an Understand project with imported OSS source code by searching with the name of the Class or Package. This phase consists of three steps: for each architectural thread that discusses architecture changes, (1) extract the names of the classes and packages related to the architecture changes from the architectural thread, and the time stamp of and the version information related to the architectural thread; (2) search iteratively the names of the classes and packages in the source code of OSS releases, which is closely before and after the time stamp when the architectural thread was communicated, using Understand tool, e.g., Hibernate v0.8.1 and Hibernate v1.0; (3) the difference between two releases in source code can be used to locate and verify the changes of architecture. For example, if a new Class discussed in an architectural thread appears in a certain release, e.g., v1.0, but does not exist in a previous version, e.g., v0.8.1, we can confirm that the communication in this architectural thread caused an architecture change (i.e., adding the new Class). The releases of Hibernate and ArGoUML used for verifying architecture changes by Understand tool are shown in Table 3. Note that not all releases of Hibernate and ArGoUML were used in this study to verify architecture changes. We checked two OSS versions before and after the discussion of a thread about architecture change to judge whether the change mentioned in the thread happened or not, for example, for an architecture change discussed about in Hibernate v0.8.1, when the change had been confirmed in Hibernate v1.0, Hibernate v0.9 was omitted from checking.

Table 3. Releases of Hibernate and ArGoUML for verifying architecture changes

<table>
<thead>
<tr>
<th>OSS Project</th>
<th>Investigated Release</th>
<th>Release Date</th>
<th>OSS Project</th>
<th>Investigated Release</th>
<th>Release Date</th>
</tr>
</thead>
</table>

* https://scitools.com/
The third phase is data analysis. Qualitative and quantitative data of architecture changes are extracted from architectural threads, and used to answer the research questions. We manually checked architecture changes discussed in each architectural thread in the mailing lists and recorded the causes of architecture changes identified in the threads in an Excel spreadsheet for further analysis. The approaches for data analysis are elaborated in Section 3.4.

3.4. Data analysis

A top-down approach, which can be used to classify domain knowledge based on an existing categorization (e.g., a predefined ontology) [28], was used in this work to classify the causes of architecture changes extracted from architectural threads. With this top-down approach, we recorded the content of each cause of architecture change and mapped the content to a categorization of architecture change cause, which is the result of a systematic review on architecture change [17]. In this way, we get an understanding of the causes of architecture changes through communication in OSS mailing lists.

A bottom-up approach can be used to classify specific domain knowledge which is not predefined in existing concepts [28]. Grounded Theory (GT) [29] as a bottom-up approach is suitable for analyzing qualitative data (i.e., the detailed causes of architecture changes in this study), and is complementary to the top-down approach. We extract concepts from the communicated content of causes of architecture changes in mailing lists using the coding techniques of GT and generate classification codes of the detailed causes of architecture changes. The process of GT is described as the following: We first employ open coding to generate codes for clustered incidents into concepts and categories, then use selective coding to identify the core categories. Any disagreements on the understanding of the coding results are discussed by two authors to reduce the personal bias. Note that we did not use theoretical coding of GT. The reason is that this coding phase will conceptualize hypotheses from code relations, which is not the purpose.
of this work. GT is conducted in an iterative process. The codes are constantly improved in the GT process. The results are presented in Section 4.1.

4. Results and Discussion

4.1. Cause of architecture change

Using a top-down approach by analyzing the extracted architecture changes, we identified the categories of causes of architecture changes. Architecture changes can be classified in different perspectives. Cause of architecture change is one of them. A recent literature review specifies four categories of causes for architecture changes [17], based on the categories of software changes identified in [30]:

1) **Perfective changes** result from new or changed requirements. These changes improve the system to better meet user needs. For example, BC introduced the idea of using code generator to generate JavaBean classes and adding BasicRenderer class to write out the classes defined in the mapping files in Hibernate.

2) **Corrective changes** occur in response to defects in software products. For example, to fix the problem of stereotypes disappearing in pane, AC added a new class ItemListener to monitor the stereotype field instead of using ActionListener class in ArgoUML.

3) **Adaptive changes** occur when moving to a new environment, platform, or accommodating new standards. For example, MK advised using the POJOs technique as an alternative of the EObjects technique in Hibernate in order to support the integration of Eclipse EMF objects with Hibernate.

4) **Preventative changes** ease maintenance by restructuring/reengineering the system. For example, DC suggested refactoring the code of MappingByReflection class into a few top level classes in Hibernate, which can facilitate the usage of this class and make maintenance easier.

The extracted causes of architecture changes can be directly mapped to the abovementioned four categories of causes for architecture changes with the top-down approach (i.e., following an existing categorization provided in [17]). The percentages of the four types of causes of architecture changes in Hibernate and ArgoUML are shown in Table 4. Note that the base values used in the proportions in Table 4, Table 6, and Table 7 are the numbers of threads that leads to architecture changes in Hibernate and ArgoUML respectively.

Table 4. Proportions of the four types of causes of architecture changes in Hibernate and ArgoUML

---

1 Only the abbreviations of the developers' names are provided due to privacy concerns.
2 http://sourceforge.net/p/hibernate/mailman/message/5496885/
4 http://sourceforge.net/p/hibernate/mailman/message/12922461/
5 http://sourceforge.net/p/hibernate/mailman/message/5496419/
As mentioned in Section 3.4, we further analyzed the causes of architecture changes using GT, which codes the concepts of the causes to three categories with fourteen codes: functional requirement (i.e., architecture changes to meet the functional requirements of a system), external quality requirement (i.e., architecture changes to meet the quality characteristics of a system that face its users), and internal quality requirement (i.e., architecture changes to meet the quality characteristics of a system that concern developers) [31]. Table 5 provides the description of the codes. The percentages of each code (i.e., subtype of architecture change cause) are shown in Table 6 and Table 7 for Hibernate and ArgoUML respectively.

Table 5. Coding results of the causes of architecture changes using grounded theory

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>New component</td>
<td>Adding a component in the system to meet a functional requirement, e.g., adding a new component for selecting a Transaction class in Hibernate.</td>
<td>Functional requirement</td>
</tr>
<tr>
<td>UI adjustment</td>
<td>Adjusting user interface of the system to meet a functional requirement, e.g., adding a tree component to provide a description field in a split pane.</td>
<td>External quality requirement</td>
</tr>
<tr>
<td>Customization</td>
<td>Meeting to customized requirements from various users, e.g., the design of the code generators in ArgoUML allows users to write their own tools for generating code.</td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td>Improving the performance of the system with available resources, e.g., achieving the required response time with limited memory.</td>
<td></td>
</tr>
<tr>
<td>Extensibility</td>
<td>The system can be extended to accommodate new environments, e.g., other systems and techniques.</td>
<td></td>
</tr>
<tr>
<td>Localization</td>
<td>Adapting the system to fit special countries/regions, e.g., adding German support.</td>
<td></td>
</tr>
<tr>
<td>Conciseness</td>
<td>Making the architecture elements (e.g., classes/packages/components) simple and readable (e.g., through architecture refactoring).</td>
<td></td>
</tr>
<tr>
<td>Consistency</td>
<td>The design elements should be consistent with each other, e.g., maintaining the design decision of separating GEF (Graphical Editing Framework) from ArgoUML.</td>
<td>Internal quality requirement</td>
</tr>
<tr>
<td>Compatibility</td>
<td>New components should be compatible with existing components/systems/technologies, e.g., a new component for initial processing in code generator should be compatible with the existing mapping file and database in Hibernate.</td>
<td></td>
</tr>
<tr>
<td>Completeness</td>
<td>The architecture design of the system can address all the</td>
<td></td>
</tr>
</tbody>
</table>
requirements in the specification.

**Flexibility**
The architecture of the system can be easily adapted to address new requirements, e.g., ArgoUML introduces a Tree component which provides the ability to easily add new functionality.

**Modularity**
The degree to which a system is composed of discrete components such that a change to one component has a minimal impact on other components.

**Traceability**
The ability of tracing from one design element (e.g., class/package/component) to another.

**Understandability**
The degree of how easy developers can understand the design of the system.

The results in Table 6 and Table 7 show that “New component” plays a dominant role in perfective architecture changes. The main causes of preventative architecture changes in Hibernate and ArgoUML are different: “Conciseness” vs. “Modularity”. The three most frequently-communicated internal qualities requirements that lead to architecture changes are “Conciseness”, “Consistency”, and “Modularity” in Hibernate and “Modularity”, “Understandability”, and “Conciseness” in ArgoUML. The two most frequently-communicated external qualities requirements that result in architecture changes are “Extensibility” and “Efficiency” in both Hibernate and ArgoUML.

The proportions of the four types of causes of architecture changes before and after the first stable release in Hibernate and ArgoUML are shown in Table 8. The abbreviations BFR and AFR represent two stages as described in the rationale of RQ2, i.e., before and after the first stable version was released. Note that the sum of the percentages of ArgoUML shown in Table 4, Table 6, Table 7, and Table 8 exceeds 100%, because one architecture change may be caused by several types of reasons (e.g., adaptive change and perfective change).

**Answer to RQ1:** There are four types of causes of architecture changes in OSS development: perfective changes, corrective changes, adaptive changes, and preventative changes, which cover all the types of causes of architecture changes in [17]. As shown in Table 4, the major cause for architecture changes in both Hibernate and ArgoUML is preventative changes. As shown in Table 6 and Table 7, the causes of architecture changes are further classified to functional requirement, external quality requirement, and internal quality requirement using the coding techniques of GT.

**Answer to RQ2:** Perfective changes and preventative changes are the main causes of architecture changes before the first stable release in both Hibernate and ArgoUML. As shown in Table 8, after the first stable version was released, the causes of architecture changes of Hibernate are mixed, e.g., adapted to JDK5 (adaptive change) and redesigning Hibernate to be more event-oriented (preventative change); the causes of architecture changes of ArgoUML are mostly perfective changes, e.g., adding a data interface for a new component.

The results in Table 6 and Table 7 show that “New component” plays a dominant role in perfective architecture changes. The main causes of preventative architecture changes in Hibernate and ArgoUML are different: “Conciseness” vs. “Modularity”. The three most frequently-communicated internal qualities requirements that lead to architecture changes are “Conciseness”, “Consistency”, and “Modularity” in Hibernate and “Modularity”, “Understandability”, and “Conciseness” in ArgoUML. The two most frequently-communicated external qualities requirements that result in architecture changes are “Extensibility” and “Efficiency” in both Hibernate and ArgoUML.
### Table 6. Proportions of the subtypes of architecture change cause in Hibernate

<table>
<thead>
<tr>
<th></th>
<th>Functional Requirements</th>
<th>External Quality Requirements</th>
<th>Internal Quality Requirements</th>
<th>Perfective Changes</th>
<th>Corrective Changes</th>
<th>Adaptive Changes</th>
<th>Preventative Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>New component</td>
<td>17.1%</td>
<td>Extensibility 5.7%</td>
<td>Consistency 11.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UI adjustment</td>
<td>2.8%</td>
<td>Extensibility 5.7%</td>
<td>Consistency 11.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.8%</td>
<td>Conciseness 5.4%</td>
<td>Consistency 5.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td>2.8%</td>
<td>Efficiency 2.9%</td>
<td>Efficiency 2.9%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Understandability</td>
<td>2.8%</td>
<td>Extensibility 9.0%</td>
<td>Extensibility 9.0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completeness</td>
<td>2.8%</td>
<td>Concurrency 3.0%</td>
<td>Concurrency 3.0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 7. Proportions of the subtypes of architecture change cause in ArgoUML

<table>
<thead>
<tr>
<th></th>
<th>Functional Requirements</th>
<th>External Quality Requirements</th>
<th>Internal Quality Requirements</th>
<th>Perfective Changes</th>
<th>Corrective Changes</th>
<th>Adaptive Changes</th>
<th>Preventative Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>New component</td>
<td>18.8%</td>
<td>Extensibility 5.4%</td>
<td>Consistency 5.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UI adjustment</td>
<td>13.5%</td>
<td>Extensibility 5.4%</td>
<td>Consistency 5.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.7%</td>
<td>Extensibility 2.8%</td>
<td>Extensibility 2.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td>2.7%</td>
<td>Modularity 5.4%</td>
<td>Modularity 5.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Understandability</td>
<td>2.7%</td>
<td>Efficiency 3.0%</td>
<td>Efficiency 3.0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completeness</td>
<td>2.7%</td>
<td>Consistency 3.0%</td>
<td>Consistency 3.0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Note: The tables show the proportions of architecture change causes in Hibernate and ArgoUML, categorized by perfective, corrective, adaptive, and preventative changes, with specific subtypes such as new component, UI adjustment, and efficiency.*
Table 8. Proportions of the four types of causes of architecture changes before and after the first stable release in Hibernate and ArgoUML

<table>
<thead>
<tr>
<th>OSS Project</th>
<th>Perfective Changes</th>
<th>Corrective Changes</th>
<th>Adaptive Changes</th>
<th>Preventative Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFR Hibernate v1.0</td>
<td>17.1%</td>
<td>5.7%</td>
<td>0.0%</td>
<td>22.9%</td>
</tr>
<tr>
<td>AFR Hibernate v1.0</td>
<td>8.6%</td>
<td>0.0%</td>
<td>20.0%</td>
<td>25.7%</td>
</tr>
<tr>
<td>BFR ArgoUML v0.10</td>
<td>35.1%</td>
<td>2.7%</td>
<td>0.0%</td>
<td>43.2%</td>
</tr>
<tr>
<td>AFR ArgoUML v0.10</td>
<td>8.1%</td>
<td>2.7%</td>
<td>0.0%</td>
<td>2.7%</td>
</tr>
</tbody>
</table>

4.2. Discussion of study results

Categorization of causes of architecture changes: All the causes of architecture changes in Hibernate and ArgoUML can be mapped to the four categories of causes of architecture changes provided in [17] and no new category was identified, which empirically validates that this existing categorization does work with OSS projects. As shown in Table 6 and Table 7, “New component” plays a dominant role in perfective architecture changes. It is a reasonable result in that new features request and introduce new components that lead to architecture changes. The main identified causes of preventive architecture changes are “Conciseness” in Hibernate and “Modularity” in ArgoUML. The reasons are the following: Hibernate was extended from a core system to several subsystems, e.g., Hibernate OGM and Hibernate Search from version 1.2.5 to version 2.0. The architecture of Hibernate needs to evolve constantly to accommodate the introduction of new subsystems, since no predefined architecture is planned and available. Developers intend to make the architecture elements (e.g., classes/packages/components) simple and readable (e.g., through architecture refactoring) to easy the understanding of the modified architecture. On the other hand, ArgoUML follows a mature architecture pattern in the whole development lifecycle: MVC (Model-View-Controller). The main goal of architecture design in ArgoUML is “to build a modular, extendable structure”, and consequently “Modularity” is the most concerned internal quality requirement when changing the architecture of ArgoUML. Note that we did not analyze corrective and adaptive architecture changes based on the coding results of GT, because their extracted data are limited, e.g., only two adaptive changes were identified in ArgoUML.

Stable and maintainable architecture: As shown in Table 8, 45.7% architecture changes in Hibernate and 89.1% architecture changes in ArgoUML happened before the first stable version. It implies that the major part of the ArgoUML architecture was formed and became stable in its initial stage, which is not the case for the Hibernate architecture. The reasons are the following: ArgoUML decided to employ MVC architecture pattern at the early stage, and the MVC pattern can well support the extension of ArgoUML after its first stable release was published. Hibernate evolved from a core system into several subsystems after its first stable release was published, and

---

k http://sourceforge.net/p/hibernate/mailman/message/7781360/
1 http://argouml.tigris.org/ds/viewMessage.do?dsForumId=450&viewType=browseAll&dsMessageId=116285
m http://argouml.tigris.org/ds/viewMessage.do?dsForumId=450&viewType=browseAll&dsMessageId=116987
its architecture was constantly modified to accommodate the new subsystems as discussed in the previous paragraph. As illustrated in Table 4, preventative changes are the major cause for architecture changes in both projects. It is not a surprising result. Preventative changes are made to easy future maintenance and evolution. OSS developers tend to make preventative changes (anticipation) in order to achieve a maintainable and evolvable architecture (e.g., refactoring architecture design to be prepared for new or changed requirements). Corrective and adaptive changes are in a small proportion in all architecture changes. The reasons are diverse, potential defects and environmental changes can be prevented and mitigated through preventative changes, or corrective changes are communicated in other sources (e.g., JIRA).

Role of core developers: Core developers refer to those that are actively involved in high levels of communication and knowledge sharing in development [32] (i.e., architecture information communication in this work), e.g., GK in Hibernate and AC in ArgoUML. In this study, we find that 68.5% architecture changes are made by the top two core developers in Hibernate, and 75.7% architecture changes are conducted by the top three core developers in ArgoUML, according to the core developers identified in [14]. These results show that core developers dominate the changes of architecture. These core developers also act as the role of architect in the two OSS projects.

4.3. Implications for researchers and practitioners

For researchers: The results of this study empirically show that architecture communication in OSS mailing lists is correlated with architecture changes in source code. One of the merits of the architecture information communicated in mailing lists is that it contains rich design rationale information about architecture (e.g., cause information about architecture changes), which is particularly useful to enrich architectural design decisions [33] and architecture documentation [34]. Another promising benefit of architecture changes classification in a cause perspective is that it allows researchers to develop a common approach to deal with the changes with similar causes, instead of addressing each change individually (e.g., the purpose of requirements changes classification [35]). To support the approach, a tool for addressing various types of architecture changes can be developed, e.g., certain architecture changes are better addressed by resolving their conflicts with related design decisions [20].

For practitioners: Participants of OSS projects may use the study results to guide them to trace architecture changes from mailing lists. As we have discussed in Section 4.2, architecture changes frequently happened before the first stable version was released. For example, if a new developer of an OSS project wants to get the basic knowledge about the architecture design in order to have a preliminary understanding of the system, s/he can check the architectural threads that suggest, negotiate design, and interpret design implementation through the mailing list during the early stage of development before the first stable release.
5. Threats to Validity

The threats to the validity of this study are discussed according to the guidelines in [36].

**Construct validity** in this study focuses on whether we extracted architecture information from the mailing lists, identified architecture changes, and interpreted the results of this study correctly. To mitigate the bias on architecture information definition, we chose the architectural description model in IEEE 1471-2000 standard [27] as a benchmark model to identify the threads that contain architecture information in mailing lists. Manually identifying architecture changes, and extracting and reviewing the causes of architecture changes, can improve the correctness of the extracted data and alleviate the threat of incorrect identification results by automated approaches. To identify architecture changes, we compared the changed Classes/Packages between two releases in source code using Understand tool, which mitigates the bias on confirming architecture changes. There is a risk that the results of this study might be affected by the interpretation of the criteria for extracting and identifying architecture information and architecture changes by different researchers. A pilot data extraction was conducted by two researchers to mitigate the bias on understanding and identifying architecture changes communicated in the mailing lists. Some causes of architecture changes (e.g., at the system level) might not contain all the information about architectural changes or might be discussed ambiguously, imprecisely, and vaguely. These causes of architectural changes are difficult to understand and be verified in source code by the identification method used in this study. This threat can be mitigated with an understanding of the code structure through the communication with core developers (architects).

**Internal validity** focuses on the avoidance of confounding factors that may influence the interpretation of the results of a study. There is a risk that the scope of architecture changes might be affected by the identification method used in this work. To mitigate this potential issue, we used the changed Classes/Packages to identify the architecture changes in source code by Understand tool. Some architecture changes at the system level discussed in architectural threads were excluded from analysis, because they are too abstract and we could not verify them in source code. We employed a descriptive statistics method to present the results of this study, and the threats to internal validity are minimized. The coding process of GT was conducted collaboratively by two authors, and any conflicts on the results were discussed and resolved to mitigate the threats due to personal bias. We did not intend to establish any causal relationship between architecture changes and other aspects (e.g., change time) of OSS development in this study.

**External validity** refers to the degree to which our findings from this study can be generalized. In order to improve the generalizability of the study results and findings, we chose two popular and representative OSS projects that have mailing lists based on the selection criteria in Section 3.2. Studying the mailing lists of more OSS projects based on the selection criteria can also alleviate this threat.
Reliability focuses on whether the study yields the same results if other researchers replicate it, which in this work is related to the collection and analysis of architecture changes as well as their causes. By making explicit the process and criteria of data collection and data analysis of this study in Section 3, and using Understand (a third-party tool) for verifying architecture changes, this threat is mitigated.

6. Conclusion and Future Work

In this study, we analyzed the architecture information communicated in architectural threads of the mailing lists of two popular OSS projects: Hibernate and ArgoUML, and located and verified architecture changes by comparing the differences between the source code of OSS releases. Four types of architecture changes in a cause perspective are classified. The main findings of this work are: (1) architecture information communicated in OSS mailing lists does lead to architecture changes in code; (2) the major cause for architecture changes in both Hibernate and ArgoUML is preventative changes, and the causes of architecture changes are further classified to functional requirement, external quality requirement, and internal quality requirement using the coding techniques of grounded theory; (3) more than 45% of architecture changes in both projects happened before the first stable version was released.

The results of this study provide promising research directions: (1) The results and findings of this work can be further validated through a survey with the core developers of the two OSS projects; (2) As we mentioned in Section 4.3, a tool for a certain type of causes of architecture changes can be developed to deal with similar architecture changes based on the results of this work; (3) Other sources in OSS development, e.g., forums, commit data [37], and blogs [38], may also contain information about architecture changes and their causes. We may explore the possibility to identify architecture changes and their causes from these sources for projects of various domains and languages (e.g., C#). (4) Topic model is a method for finding patterns of words in document collections using hierarchical probabilistic model in the field of natural language processing (NLP) [39]. We can employ this method to automatically analyze the topics of selected architectural posts, and compare the results to the causes of architecture changes manually identified in this work.

Acknowledgments

This work is partially sponsored by the NSFC under Grant No. 61170025 and 61472286. Thank the SEKE 2015 and IJSEKE reviewers for their valuable comments to this work.

References

18 W. Ding, P. Liang, A. Tang & H. van Vliet


