Assumptions in OSS Development: An Exploratory Study through the Hibernate Developer Mailing List

Zhuang Xiong and Peng Liang*
School of Computer Science
Wuhan University
Wuhan, China
{zxiong, liangp}@whu.edu.cn

Chen Yang
IBO Technology (Shenzhen) Co., Ltd.
Shenzhen
China
c.yang@ibotech.com.cn

Tianqing Liu
School of Computer Science
Wuhan University
Wuhan, China
tqliu@whu.edu.cn

***Abstract*—Developers constantly make various assumptions regarding requirements, environment, design decisions, etc. during software development. However, these assumptions are usually implicit and undocumented and there is a lack of understanding regarding what assumptions have been made and discussed in software development. Open Source Software (OSS) is recently becoming an important part of software industry. To this end, we conducted an exploratory study on assumptions in OSS development. We extracted and analyzed 9006 posts from the developer mailing list of Hibernate (a popular OSS project), in order to explore (1) assumption expression and (2) classification, (3) the trend of assumptions over time, and (4) related software artifacts of assumptions in OSS development. We identified 832 assumptions from the Hibernate developer mailing list. The findings are: (1) most of the assumptions are expressed as "Feature Request" and "Solution Proposal"; (2) more than half of the identified assumptions are design assumptions and are made for software design; (3) assumptions exist in the whole OSS development lifecycle; and (4) the major category of related artifacts of assumptions is "Design Document".**

Keywords*—*Assumption, Open Source Software, OSS Development, Hibernate, Mailing List

I. INTRODUCTION

From a software engineering perspective, the assumption concept is not new. Lewis *et al*. explored assumption management as a method for improving software quality, and they declared that stakeholders such as developers, architects, and maintainers constantly make assumptions during their daily work for different purposes [1]. Lago and van Vliet elaborated how software evolution can be hampered by implicit assumptions. They also presented a metamodel for Assumption Documentation [2]. Moreover, Tirumala *et al.* found that an important concern in real time system development is the issues caused by not well-managed assumptions [3].

Assumptions have a long history in software development, and many works have pointed out the importance of assumptions. For example, Garlan *et al*. declared that incompatible assumptions may result in architectural mismatch in software architecture [4]. Furthermore, developers may misunderstand assumptions in software development, which could lead to misunderstandings in other software artifacts related to such assumptions [5]. Another example is from Steingruebl and Peterson's argument. They proposed that undocumented software assumptions might cause the failure during software development, and they also suggested methods to manage assumptions [6]. In our recent systematic mapping study [7], we argued that assumptions are a type of specific software development knowledge that stakeholders believe but are not sure regarding their correctness, impact, importance, etc., and we found that assumptions and their management span the whole lifecycle of and are critical in software development.

On the other hand, OSS is recently becoming an important part of software industry [8]. Though there are some works regarding assumptions and their management either in general (e.g., considering assumption management that contributes to software quality [1]), or in a specific development phase (e.g., requirements engineering or architecting) [7], we did not find evidence regarding assumptions and their management in OSS development. To this end, we conducted an exploratory study on assumptions in OSS development. More specifically, we analyzed the posts from the developer mailing list of Hibernate, in order to explore (1) assumption expression and (2) classification, (3) the trend of assumptions over time, and (4) related software artifacts of assumptions in OSS development. The major reasons for choosing Hibernate and mailing list are: (1) Hibernate is a popular OSS project that has a wide range of applications in database systems, and (2) communications in OSS development are typically conducted in an open and public manner [9]. Mailing lists play an important role for knowledge sharing between knowledge providers and seekers [10]. As many developers in OSS projects use mailing lists for communication [11], those mailing lists have been a valuable source for researchers to analyze and understand OSS development practices. For example: Bird *et al*. proposed that mailing lists could be used for investigating handing of patches [12]; Mockus *et al*. also argued that mailing lists could help with quantifying defect density [13]; Finally, Bird *et al*. declared that mailing lists provide a useful trace of the communication and co-ordination activities of the developers [9].

The rest of this paper is structured as follows: Section II provides related work. Section III describes the research method in detail. Section IV presents the results of the study, which are further discussed in Section V. The threats to validity are described in Section VI. Section VII concludes this work with future directions.

II. RELATED WORK

A. Assumptions in Software Engineering

Many researchers and practitioners have pointed out the significance of assumptions in software development. Lehman revealed that invalidation of assumptions in software development may lead to disastrous failures in software systems [14]. A typical example is the reuse of an assumption from ARIANE 4, which was invalid in ARIANE 5 and resulted in the ARIANE 5 failure [15]. Haly

* Corresponding author

978-1-7281-1970-0/18/$31.00 ©2018 IEEE
DOI 10.1109/APSEC.2018.00060

et al. presented that requirements engineers can use trust assumptions to help ensure the scope of analysis and the decisions documentation [16]. Lewis *et al.* identified several problems caused by implicit assumptions in software projects. For example, if assumptions are implicit, stakeholders may not be aware of the evolution of these assumptions [1]. Subsequently, some assumptions may turn out to be invalid over time, which may result in unexpected consequences. Lago and van Vliet noted that stakeholders make assumptions regarding what will change during development, and they connected those assumptions to system variability and invariability [2]. Additionally, they designed a metamodel for Assumption Documentation, and showed how to organize assumptions in an explicit way based on a product family. Furthermore, in our previous work, we systematically investigated and surveyed the literatures published between 2001 and 2015, and provided practitioners and researchers with a reflection of the exploration and analysis on assumptions and their management in software development [7]. There are also many other works related to assumptions and their management. However, to the best of our knowledge, there is no work that aims to investigate assumptions and their management in OSS development. This work is the first step to close this gap.

There are many kinds of classifications towards assumptions in software developments. Lewis *et al*. designed a prototype tool called the Assumption Management System (AMS) which is expected to improve software quality. AMS records and extracts assumptions from Java source code into a repository and manages these assumptions in a web-based format. These assumptions were characterized into five types [1]: Control Assumption, Environment Assumption, Data Assumption, Convention Assumption, and Usage Assumption. The categories above describe the taxonomy used to group the assumptions gathered from source code. This research specifically focuses on the development of a classification of assumptions during coding phase. Garlan *et al*. proposed four types of assumptions that can be conducive to architectural mismatch: Nature of components, Nature of the connectors, Global architectural structure, and Construction process [4]. Moreover, Lago and van Vliet divided architectural assumptions into Technical Assumptions, Organizational Assumptions, and Managerial Assumptions [2]. As far as we know, there is no general classification of assumptions in software development, and we plan to first explore the classification of various assumptions in the context of OSS development.

B. Software Artifacts in OSS Development

Developers aim to improve the quality of information captured in different forms of software artifacts such as requirement documents, bug reports, source code during software development. These software artifacts may contain excessive information, which is difficult to comprehend [18]. Fernandez *et al*. declared that software artifacts play an important role in systems and software development processes, and they concluded that an artifact is a self-contained work result, which constitutes a physical representation, a semantic content, and a syntactic structure [18]. Furthermore, Ma and Fakhoury identified the types of software artifacts produced by a wide variety of OSS projects at different levels of granularity. They also presented an automated approach to identify different types of software artifacts [19]. In our recent systematic mapping study [7], we explored software artifacts related to assumptions in software development, and identified different types of relationships between them. In this work, we aimed to understand which artifacts are related to assumptions in the context of OSS development based on both the classification proposed in [19] and [7].

III. Study Design

This case study is exploratory [17], because it aims to identify new insights of assumptions in OSS development, which can support further research on this topic (e.g., generation of new ideas for assumption management).

A. Goal and Research Questions

The goal of the case study is to **analyze** the developer mailing list of Hibernate **for the purpose of** characterization **with respect to** assumption expression and classification, the trend of assumptions over time, and related software artifacts **from the point of view of** OSS developers **in the context of** OSS development.

RQ1: How do developers express assumptions in OSS development?

Rationale: Software developers constantly make assumptions (e.g., when analyzing requirements or making design decisions) in their daily work [1]. However, there is a lack of evidence regarding which terms and patterns are used to express assumptions in OSS development. This RQ aims to fill the gap.

RQ2: Which types of assumptions are made in OSS development?

Rationale: Though certain classifications of software assumptions have been proposed in literature (e.g., [2]), those classifications are usually limited to a specific scope (e.g., architectural assumptions). The aim of this RQ is to provide an overview of the assumptions made in OSS development.

RQ3: What is the trend of assumptions made in OSS development over time?

Rationale: There is a lack of evidence regarding the variation of assumptions (e.g., number) over time in OSS development [21]. The results of this RQ help to gain knowledge of such trend, which can further support assumption management (e.g., identifying the best timing to manage assumptions).

RQ4: Which types of software artifacts are related to assumptions made in OSS development?

Rationale: Assumptions are not independent in software development, but intertwined with certain types of software artifacts [7]. However, there is a lack of evidence regarding which types of artifacts are related to assumptions in OSS development. This RQ aims to fill the gap.

B. Case Selection

Hibernate provides a framework for mapping an object-oriented domain model to a relational database in Java, which is widely used and popularized in the world. Hibernate has 9006 posts in its developer mailing list

between 2002 and 2015, which offers abundant data about the communication of development knowledge between developers. Therefore, we decided to choose the Hibernate developer mailing list to perform our case study. The detail information of the Hibernate developer mailing list is provided in TABLE I.

TABLE I. DEVELOPER MAILING LIST OF HIBERNATE

Project	Mailing List URL	Time Period	Number of Posts
Hibernate	http://sourceforge.net/p/hibernate/mailman/hibernate-devel/	Jan 2002 - Jun 2015	9006

C. Study Procedure

The case study procedure is composed of the following steps:

Step 1: The first and fourth author read the Hibernate developer mailing list to identify and extract assumptions in parallel based on the guidelines in Section III.D;

Step 2: The first and fourth author reviewed the identified assumptions;

Step 3: The first and fourth author analyzed the identified assumptions;

Step 4: The second and third author reviewed the results of analysis with the first and fourth author in order to address problems and conflicts.

The data items to be extracted are mapped to the RQs as shown in TABLE II in the Appendix. Note that we used a predefined list of terms for identifying assumptions and answering RQ1: *assume*, *assuming*, *assumption*, *guess*, *plan*, *suppose*, *need*, *will*, *would*, *might*, *could*, *seem*, *probably*, *maybe*, *possibly*, *perhaps*. TABLE IX in the Appendix provides a description for each term according to Oxford English Dictionary [22]. This list was iteratively maintained during data extraction and analysis.

TABLE II. DATA ITEMS TO BE EXTRACTED FROM THE POSTS

#	Data item	Description	RQ
D1	Terms	Terms describing assumptions that are extracted from the post.	RQ1
D2	Linguistic Patterns	Linguistic patterns describing assumptions that are extracted from the post.	RQ1
D3	Content	The main content of the post related to assumptions from the post.	RQ2
D4	Release	In which release the post was communicated.	RQ3
D5	Related Artifacts	Software artifacts that are related to assumptions in the post.	RQ4
D6	Relationships	Relationships between assumptions and their related artifacts	RQ4

D. Data Collection

Most of previous works do not define the word "assumption"; instead they directly use this word in their definitions of "assumption" [7]. To make the collection of "assumptions" in our work explicit, we use Boolean "AND" to join each of the following guidelines to collect data.

(1) Every assumption should be characterized by uncertainty; if something is certain or has strong evidence supporting its validity, for example, if stakeholders have enough evidence about the importance, impact, or correctness of specific software development knowledge, it is not an assumption. Despite this inherent uncertainty, an assumption is still taken for granted or accepted as true. Therefore, if such characteristic can be found in a piece of information, it increases the possibility that the information is an assumption. Please note that assumptions are not equal to uncertainties in software development, instead assumptions can result from uncertainties [7]. In other words, making explicit or implicit assumptions is a way of addressing uncertainties [2].

(2) Look at the context and not just the content. In our systematic mapping study [7], we identified many examples of assumptions from the selected studies (e.g., "*There is a subsystem that is responsible for receiving emergency calls and forwarding them to an available Coordinator*" [23]; "*If thread i holds the lock in read mode, then x cannot be changed by another thread*" [24]). We argue that it is difficult to treat such examples as assumptions by only reading these statements [7]. However, when we dig deeper, paying attention to the context of those examples, it becomes clear why the examples are considered as assumptions, instead of other types of artifacts, such as requirements.

(3) Uncertainty may also appear in other types of software artifacts, but cannot be the core of these artifacts. For instance, when discussing on a design decision, stakeholders generally pay attention to address the related problems, but do not notice whether it is uncertain. On the contrary, assumptions are often made to handle uncertainties [7].

(4) There is a difference between the content of an assumption and other types of artifacts. As an example, the content of a decision focuses on giving a solution to the problem, while the content of an assumption concerns more about something is correct, suitable, valuable, etc. [7].

E. Data Analysis

We used predefined classifications to categorize assumption expression (RQ1) and related software artifacts (RQ4), and descriptive statistics to present the trend of assumptions (RQ3). For RQ1, a recent literature reviewed the content of OSS developer mailing list and classified them into six categories [25], and we referred to this classification in our work. For RQ3, we defined three months as the unit for the statistics of the number of assumptions. For RQ4, we referred to the categories of software artifacts in [19] and the relationship types between assumptions and other types of artifacts in our systematic mapping study [7]. Moreover, a bottom-up approach is usually used for establishing a theory or model in view of existing domain knowledge [27]. We chose Constant Comparison (CC) [28] as such an approach for RQ2. CC is a systematic approach to generate concepts from extracted data, and needs a continuous process of generation and

verification [29]. We used CC to code and extract concepts from the communicated content of assumptions in mailing lists, and generate a classification of the identified assumptions. More specifically, (1) we first employed open coding to generate codes for clustered incidents into concepts and categories, (2) we further refined the categories and relations, and (3) we compared all existing concepts, and identified commonalities in those concepts to form core categories [30]. TABLE III shows the relationships among the extracted data items, data analysis methods, and research questions.

TABLE III. THE RELATIONSHIPS AMONG DATA ITEMS, DATA ANALYSIS METHODS, AND RESEARCH QUESTIONS

#	Data item	Data analysis method	RQ
D1, D2	Terms; Linguistic Patterns	Predefined Classification	RQ1
D3	Content	Constant Comparison	RQ2
D4	Release	Descriptive Statistics	RQ3
D5, D6	Related Artifacts; Relationships	Predefined Classification	RQ4

F. Pilot Study

We conducted a pilot study with 300 posts randomly selected from the 9006 posts (as described in Section III.B). The pilot study followed exactly the same procedure of the case study. Through the pilot study, we learned that it was important to provide an explanation for each data item, and should strictly deal with the bias between researchers (i.e., the first and fourth author) on identifying assumptions during the formal procedure of the case study.

IV. STUDY RESULTS

A. Results of RQ1

We identified and extracted 832 assumptions from the Hibernate developer mailing list. Note that one post may contain one or more assumptions. We extracted and categorized assumption expressions into six types of linguistic patterns. TABLE IV shows the identified types of linguistic patterns for expressing assumptions with their descriptions and examples. These six types of linguistic patterns can be used to express assumptions generally contained in the developer mailing list regarding OSS development. Also, we provide the concrete linguistic patterns for expressing assumptions collected from the Hibernate developer mailing list in TABLE X (Appendix). The extracted content related to assumptions can be directly mapped to the six types of linguistic patterns (i.e., following the categorization in [25]). The percentages of the six types of linguistic patterns of assumptions from the Hibernate developer mailing lists are also shown in TABLE IV. The major types of linguistic patterns of assumptions are "Feature Request" (40.7%) and "Solution Proposal" (39.6%), while the other types receive much less attention.

B. Results of RQ2

TABLE V shows a classification of assumptions in the Hibernate developer mailing list, and the percentage of each subtype of assumptions is presented in TABLE VI. The major category of assumptions in the Hibernate mailing list is Design Assumption, followed by Requirement Assumption. These two categories of assumptions are nearly 80% of all the identified assumptions.

TABLE IV. PERCENTAGES OF THE SIX TYPES OF LINGUISTIC PATTERNS OF ASSUMPTIONS FROM THE HIBERNATE DEVELOPER MAILING LIST

Linguistic Patterns	Description	Example	Percentage
Feature Request	Assumptions related to developers' ideas for improving the quality of products or requirements on functionalities.	*It would be mostly a 'marketing' feature and I don't like useless features bloating the codebase.*	40.7%
Solution Proposal	Assumptions related to proposing feasible solutions for problems in software development.	*It looks like you can add an object using SessionImpl.addEntity().*	39.6%
Information Giving	Assumptions related to providing information or support for other developers.	*I think ERDs would be a big help to speed the understanding of how Object-Relational mapping is handled in Hibernate.*	7.4%
Problem Discovery	Assumptions related to unexpected events.	*For the project, concurrency may not be high but data stale can be a disaster.*	7.3%
Opinion Asking	Assumptions related to requiring other developers' viewpoints towards something.	*We will probably end up writing a query for this case, but it got me thinking about mappings with specific use cases, maybe as subclasses? Any thoughts?*	2.8%
Information Seeking	Assumptions related to attempts to get information or support from other developers.	*Is it true that to use session.disconnect() may not be necessarily better, but no doubt more secure?*	2.2%

TABLE V. A CLASSIFICATION OF ASSUMPTIONS IN THE HIBERNATE DEVELOPER MAILING LIST

Type	Subtype	Description	Example
Requirement Assumption	Functional requirement assumption	Assumption about functional requirements.	*It would be very nice if we can establish exactly which subset of Hibernate's functionality is working using the MS driver. It might turn out to be a large enough subset for some people's requirements.*
	Non-functional requirement assumption	Assumption about non-functional requirements.	*I made a decision to have Hibernate always return distinct result sets. It would allow more flexibility for the user, be more predictable (clearer semantics) and simplify the implementation.*

Design Assumption	Model assumption	Assumption about models adopted in the phase of design.	*The immediate benefit of this (but perhaps the least important one) is that Hibernate now supports a <dynamic-class> mapping. An entity may now be modeled as a Map of property values, instead of as a JavaBean. This means that Hibernate can now be used for the type of problems that we used to recommend OFBiz Entity Engine.*
	Architectural assumption	Assumption about architectural elements of the expected architecture.	*I would like to make some architectural adjustments to the org.hibernate.transaction.JTATransaction implementation that would work with or without JNDI.*
	Scenario assumption	Assumption about quality attribute scenarios and usage scenarios.	*We may have two options: 1) keep the jdk15 hack I did in the build process and gain a more robust implementation (generics usage); 2) drop it a gain simplicity and unique distribution.*
Construction Assumption	Source code assumption	Assumption about making decisions on source code.	*Yes, as I've been using HibernateService with Fortress, I haven't had the need to implement the (deprecated) ThreadSafe interface (with fortress you declare the lifestyle with meta data). But as the code itself is threadsafe, adding ThreadSafe marker interface should fix the issue you are experiencing.*
	Implementation assumption	Assumption about making decisions on implementation procedures.	*I'm going to check it out as a MVC implementation possibility for my projects in the future though.*
	Annotation assumption	Assumption about annotation added to source code	*I think bean level annotation would be sufficient. I am sure that some people can come up with some sort of use case field and method level.*
Testing Assumption	Bug assumption	Assumption about checking errors of source code.	*We could have a quick look at Castor's changelog and check how many of their bugs are to do with concurrency issues. Hibernate is essentially immune to concurrency bugs because its critical code is single threaded.*
	Testing plan assumption	Assumption about making plans on testing procedures.	*The implication there is that you would effectively not be able to set up two different test tasks to run unit and integration tests separately.*

TABLE VI. PERCENTAGE OF EACH SUBTYPE OF THE IDENTIFIED ASSUMPTIONS IN THE HIBERNATE DEVELOPER MAILING LIST

Type	Requirement Assumption		Design Assumption			Construction Assumption			Testing Assumption	
Subtype	Functional requirement assumption	Non-functional requirement assumption	Model assumption	Architectural assumption	Scenario assumption	Source code assumption	Implementation assumption	Annotation assumption	Bug assumption	Testing plan assumption
Percentage	21.7%	10.4%	10.6%	35.1%	8.7%	5.1%	3.6%	0.4%	2.8%	1.6%
	32.1%		54.4%			8.7%			4.8%	

C. *Results of RQ3*

We use a histogram (Fig. 1) to present the number of assumptions that were identified per three months during the time period of development, and mark the release versions of Hibernate (using red dot) and month information in the histogram. Note that the Hibernate developer mailing list was moved from Sourceforge to JBoss by August 2006 [31], and there are very few posts after August 2006, as shown in Fig. 1. The results in Fig. 1 show that the number of assumptions had been continuously increasing before version 2.0. Especially, a large number of assumptions had been made between version 1.0 and version 2.0. After version 2.0, the number of assumptions has a tendency of decrease, but we can see that assumptions were discussed in the whole development lifecycle (the investigated period) of Hibernate.

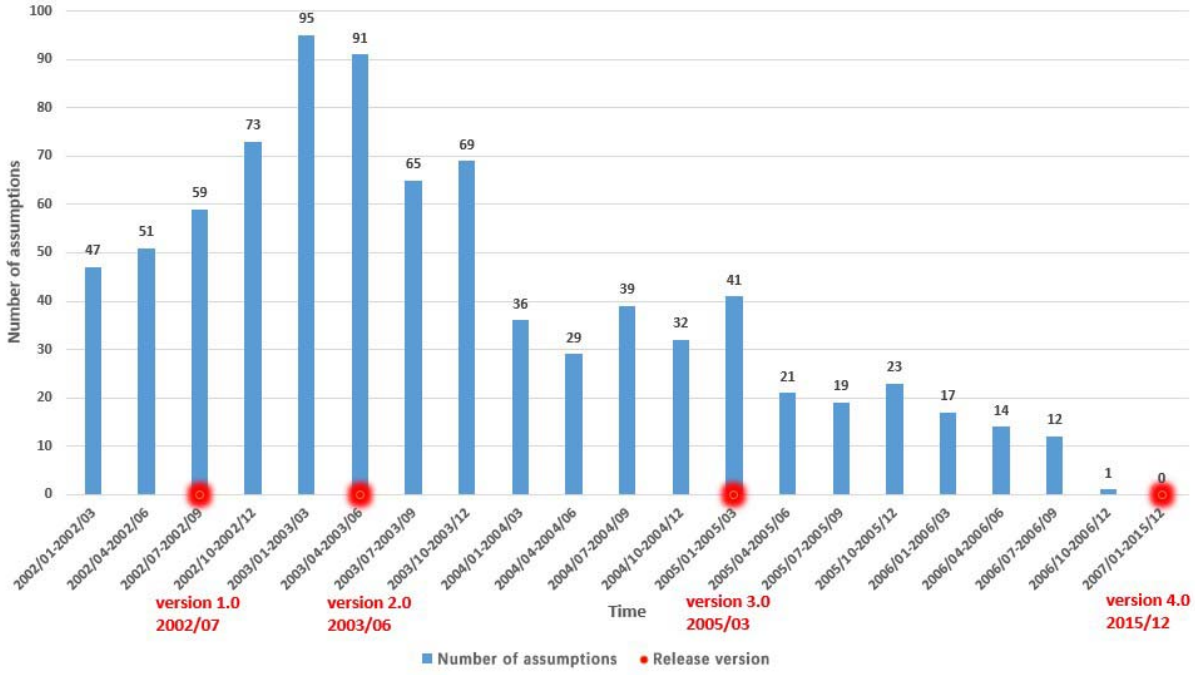

Fig. 1. Number of assumptions over the investigated period of the Hibernate developer mailing list

D. Results of RQ4

As elaborated in Section III.E, through using a predefined categorization of software artifacts [19], we classified four types of software artifacts related to assumptions, and presented the percentages of these four artifacts in TABLE VII. As shown in TABLE VII, "Design Document" (47.3%) and "Requirements Document" (38.4%) are the top two categories of artifacts related to assumptions in OSS development. We did not identify any new type of artifacts related to the identified assumptions.

TABLE VII. PERCENTAGES OF SOFTWARE ARTIFACTS CATEGORIES RELATED TO ASSUMPTIONS ON HIBERNATE

Category	Description	Percentage
Design Document	Design documents include design patterns, design models, and design decisions.	47.3%
Requirements Document	Requirements documents include functional and non-functional requirements, features, use cases, use feedbacks, and requirements decisions and models.	38.4%
Release Note	Release notes include version changes, bug fixes, and improvements regarding the project.	10.1%
Setup File	Setup files include manifest files, configuration files, and version requirements files.	4.2%

Moreover, according to our recent mapping study [7], we further identified how artifacts were related to assumptions in OSS development, as shown in TABLE VIII (A represents assumptions and B represents other types of artifacts). We provide four examples regarding how the four types of artifacts were related to assumptions. As for design documents, a developer ever mentioned: "*Your domain model classes should implement the business rules and functions. Some of them might be persistent, but that's what you have Hibernate for. Sometimes you might use a "Manager" type class in your domain model, but most of the time the objects _themselves_ interact by calling business methods on each other and/or passing themselves as arguments. You could read up on 'Strategy Pattern' and other techniques how to implement this. A good source could be Martin Fowler's Patterns of Enterprise Architecture, but it's not 'only' about domain model/transaction scripts/business logic design issues*". In this example, we can easily identify the relationship that this assumption (*design assumption*) was made for a design pattern (*strategy pattern*). As for requirements documents, a developer mentioned: "*Dynamic mapping could be a requirement for some developer and programmatic change the mapping could be useful for them. And finally It could be useful for develop a Hibernate IDE as a plugin for eclipse or other IDEs*". We understood that the assumption (*requirement assumption*) was based on a functional requirement (*dynamic mapping*). As for release notes, a developer said: "*I think it makes sense to pull the datastore-specific package up one level, moving all the types of a backend under one shared super-package. I'd like to address before doing the next OGM release is the structure of public packages in the datastore-specific modules*". We can see that this assumption (*construction assumption*) led to the change in release notes (*next OGM release*). As for setup files, as a developer presented: "*I've already suggested on the user forum that it would be great if Hibernate could lazily load a mapping file whenever it encounters a persistent class that is not yet mapped - this would eliminate the need to specify anything for the configuration*". The assumption (*construction assumption*) led to the changes in a configuration file. We can see that most of the identified assumptions were made for design documents (28.3%), and were based on requirements documents (28.1%).

TABLE VIII. PERCENTAGES OF RELATIONSHIPS BETWEEN ASSUMPTIONS AND OTHER TYPES OF ARTIFACTS

Relationship	Design Document	Release Note	Requirements Document	Setup File
A *is made for* B	28.3%	2.4%	7.2%	
A *is based on* B	5.1%		28.1%	
A *is contained in* B				
A *is* B *per se*				
A *leads to the problems in* B	2.1%			
A *leads to the changes in* B	7.6%	6.5%		3.1%
A *helps to manage* B				
A *'s failure negatively impacts* B	4.2%			
A *is related to* B		1.2%	3.1%	1.1%

V. DISCUSSION

Assumption expressions in OSS development: The results of RQ1 show that all the linguistic patterns of assumptions identified from Hibernate can be mapped to the six linguistic patterns proposed in [25] as shown in TABLE IV (i.e., no new patterns were found). This provides evidence regarding applying such a classification of assumption expressions in other OSS sources. Many literature (e.g., [2][5][21][32]) mentioned that it is difficult to distinguish assumptions from other types of artifacts in software development, and implicit and invalid assumptions lead to many problems in projects. Moreover, OSS development has its specific context, for example, face to face communication is limited by distance and therefore, hardly happens [10]. This makes assumption management in OSS development even more challenging. As shown in TABLE IV, the major types of linguistic patterns of assumptions are "Feature Request" and "Solution Proposal". The results indicate that stakeholders should at least pay attention to those assumptions in order to reduce misunderstandings and facilitate further analysis and evaluation. This requires stakeholders not only express those assumptions in an explicit and consistent way, but also provide enough information (e.g., properties and relations to other types of artifacts) to other stakeholders.

Assumptions made in OSS development: The results of RQ2 show that there are four categories of assumptions identified in the developer mailing list. Many studies (e.g., [7][21][32]) claimed that assumptions span the whole software development lifecycle, while the majority of assumptions are made or related to the artifacts in requirements engineering and software design. As shown in TABLE VI, more than half of the identified assumptions are design assumptions, followed by requirements assumptions. This indicates that though OSS development and other types of software development may differ in various aspects (e.g., context), the distribution of assumptions over development activities is similar. Additionally, we can see that architectural assumptions play an important role in design assumptions. One potential reason is that Hibernate was extended from a core system to several subsystems after version 1.0, but developers found that no predefined architecture was available. In order to accommodate the introduction of new subsystems, the architecture of Hibernate should continuously evolve.

The trend of assumptions in a time perspective: The results of RQ3 show that assumptions exist in the whole development lifecycle of Hibernate. Our study provide more evidence regarding the finding in [7][21] that assumptions exist in the whole software development lifecycle. As shown in Fig. 1, we can see that the majority of assumptions in the development of Hibernate were made before version 2.0. The main reason is that assumptions mainly were made in the early phases of OSS development (e.g., requirements engineering), due to the lack of information, evidence, and knowledge in these phases. From version 2.0 to version 4.0, the number of assumptions has decreased. It is a reasonable result because software systems (e.g., Hibernate) tend to become more and more stable. In conclusion, there are many uncertainties in the initial stage of Hibernate development, and then the system gradually resolves the uncertainties and becomes stable with the progress of the development.

Software artifacts related to assumptions: The results of RQ4 show that the artifacts related to assumptions can be divided into four categories. Many studies (e.g., [5][7][32]) presented that assumptions are not independent in OSS development, but intertwined with many other types of software artifacts. As shown in TABLE VII, the main category of artifacts related to assumptions discussed in the Hibernate developer mailing list are design documents, followed by requirements documents. Furthermore, according to the relationships presented in TABLE VIII, we can see that assumptions and the other four types of artifacts are interweaved in OSS development, and are also characterized by a set of relationships. The relationships between an assumption and another type of artifact can be multiple, and most of the identified assumptions were *made for* design documents, and were *based on* requirements documents. This indicates that developers usually make assumptions during two important development activities: requirements engineering and software design. This result is in line with the finding of our recent systematic mapping study [7] that "*most assumptions are made for or related to the artifacts in requirements engineering and software design*" and this result is also explainable because the key requirements and design decisions can be changed or introduced between versions in OSS development.

VI. THREATS TO VALIDITY

We discuss the threats to the validity of the study according to the guidelines proposed in [20].

A. Construct Validity

Construct validity reflects to what extent the research questions and the studied operational measures are consistent [20]. A potential threat is that whether the collected data can answer the RQs, since the study might be affected by, for example, different understandings among the researchers regarding the data collection guidelines. To reduce the threat, we iteratively refined the study design (e.g., RQs and data collection procedure), and we also conducted a pilot study by two researchers to refine the case study.

B. Internal Validity

Internal validity refers to the extent to which the treatment was actually responsible for the results, and unknown factors may have had an influence on the results. In this study, there is a risk that we only analyzed the

Hibernate developer mailing list at Sourceforge, while the Hibernate developer mailing list was moved from Sourceforge to JBoss by August 2006, and very limited posts were discussed after August 2006 (see Fig. 1). We are confident about the results for the investigated period from January 2002 to July 2006. After that period, the results may be affected by incomplete analysis of the posts of the Hibernate developer mailing list at JBoss. Further analyzing the posts after August 2006 can mitigate this threat.

C. External Validity

External validity is concerned with the generalization of the case study results [20]. We only investigated one OSS project (i.e., Hibernate) in this study, and our results may not be generalized to other context (e.g., projects). As we mentioned in Section I and III, we argue that Hibernate is representative to a large extent in OSS development. Therefore, the results of the case study can be generalized to similar contexts. Replication of this study to various OSS projects is one way to further reduce the threat.

D. Reliability

Reliability focuses on whether the study would yield the same results when other researchers replicate it [28]. We performed a pilot study to refine the case study design, and reduced the ambiguities in the execution of the case study (e.g., before data collection, the researchers reached a mutual understanding on the procedure and guidelines). Moreover, data collection and data analysis were conducted by two researchers in parallel, and reviewed by a third researcher. Therefore, the threat to reliability was reduced.

VII. Conclusions and Future Work

We conducted an exploratory study on assumptions in OSS development. In this study, we extracted 9006 posts from the developer mailing list of Hibernate, and manually analyzed assumptions to explore assumptions expression, classification, the trend of assumptions over time, and related software artifacts of assumptions in the context of OSS development. We identified 832 assumptions from the Hibernate developer mailing list, and our results show that: (1) Most of the assumptions are expressed as "Feature Request" and "Solution Proposal"; (2) more than half of the identified assumptions are design assumptions; (3) assumptions exist in the whole OSS development lifecycle; and (4) the major category of artifacts related to assumptions on Hibernate is Design Document and most assumptions are made for software design.

Based on the results and findings of this work, our future steps are: (1) to explore the feasibility of this study in other OSS projects; (2) to conduct a questionnaire survey towards some core developers on Hibernate to validate our findings; and (3) to investigate how linguistic patterns, classification of assumptions, and related artifacts presented in this study can further contribute to identify assumptions in different environments (e.g., bug reports, blogs).

Acknowledgments

This work is partially sponsored by the NSFC under Grant No. 61472286. The authors would also like to thank Wei Ding for collecting the posts of the Hibernate developer mailing list.

Appendix

TABLE IX. Definitions of the Assumptions Identification Terms

Term	Definition
Assume (Assuming, Assumption)	Suppose to be the case, without proof.
Guess	Estimate or conclude (something) without sufficient information to be sure of being correct.
Plan	A detailed proposal for doing or achieving something.
Suppose	Think or assume that something is true or probable, but lack proof or certain knowledge.
Need	Require (something) because it is essential or very important rather than just desirable.
Will	Express the future tense.
Would	Express a conjecture, opinion, or hope.
Might	Express a possibility based on an unfulfilled condition.
Could	Used to indicate possibility.
Seem	Give the impression of being something or having a particular quality.
Probably	Almost certainly; as far as one knows or can tell.
Maybe	Used when you are not certain that something will happen or that something is true.
Possibly	In accordance with what is likely or achievable.
Perhaps	Used to express uncertainty or possibility.

TABLE X. Identified Linguistic Patterns of Assumption Expressions from the Hibernate Developer Mailing List

#	Feature Request
1	It might be wiser to [verb]
2	Perhaps it's useful to [verb]
3	Such things should be [adjective]
4	[someone] may want to [verb]
5	[something] could also be interpreted as [something]
6	[something] could be [verb] easily
7	I think it used to be
8	I assume that
9	I think it could only work with [something]
10	I think that it [verb]
11	I really feel like that is an important [something]
12	I guess [something] can/could/would [verb]
13	I guess [someone] expect [something]
14	[someone] would assume that you won't have to [verb]
15	Perhaps [something] could be added to [something]
16	I think the most [adjective] thing is [something]
17	[someone] would like to have [something]
18	It would be possible to [verb]
19	[something] could support [something]
20	It would be better if [subject + verb]
21	[something] may need to be [verb]
22	You could check [something]
23	[something] ought to be [adjective]
24	[something] would require [something]
25	[someone] would request [something]
26	[someone] should try to [verb]
27	[something] may be what [someone] need
28	It would be meaningful to [verb]

29	[someone] could offer/provide [something]
30	Could we add [something] into [something]?
31	My hope would be that
32	I was wondering if it is possible to [verb]
33	It would be convenient to [verb]
34	It might be valuable to [verb]
35	I think it would make a difference in [something]
36	I think we would need to [verb]
37	All that is needed might be [something]
38	It would be significant to have [something]
39	You may try to get [something]
40	[something] could contain [something]
41	[something] should be attached
42	[something] might be what you want
43	It's probably not hard to do [something]
44	We could implement [something] easily
45	The most effective way to do [something] might be [something]
46	[something] could be defined more understandable
47	It would be possible to change [something]
48	My reaction would be that
49	It would be available to [someone]
50	It would be comfortable to [verb]
51	[someone] would like to keep [something]
#	**Opinion Asking**
52	Should [something] be [verb]?
53	How/what about [something]?
54	Is [something] you may prefer?
55	Does that mean [someone] should [verb]?
56	[something] needs [something], right?
57	We will probably [verb], any thoughts?
#	**Problem Discovery**
58	It could be a mistake/problem
59	There may be a problem
60	[something] might be unable to [verb]
61	[something] could not be found
62	[something] may be failed
63	[something] seems to be a mistake
64	[something] could be a risk
65	[something] may not do [something]
66	An error could come from [something]
67	[something] could be missing
68	[something] may not be able to [verb]
69	This would make [something] more complex
70	This would not be a good position to [verb]
#	**Solution Proposal**
71	I'm guessing what users really need is [something]
72	I am/was thinking this is/was a good way to [verb]
73	Perhaps a more general approach would be
74	[someone] can always just not use [something]
75	I wonder about permitting the option of doing [something]
76	I was thinking [something] might be more useful than [something]
77	I'm happy to use [something]
78	I can think of [something] that might improve [something]
79	[something] could be found [somewhere]
80	[someone] would like to offer [something]
81	could allow [something] to [verb]
82	The solution might consist of [something]
83	[something] may not be available/appropriate
84	You could try to create [something]
85	[someone] should/could execute [something]
86	Our goal could provide [something]
87	[something] could be a solution
88	[something] could be solved by [verb]
89	[something] would be a good idea
90	I could [verb] [something] to fix bugs
91	A new [something] could be made
92	The easiest way I could think of is to [verb]
93	I could consider adding [something]
94	The whole idea here could be [something]
95	I assume [something]
96	You could [verb]
97	A better way would be [something]
98	[someone] could work on [something]
99	We will most likely use [something]
100	I think it makes sense to [verb]
101	I guess [something] can [verb]
102	I would suggest [something]
103	I think [something] could solve the issue nicely
104	[something] might not be a bad idea
105	I would prefer (not) to [verb]
106	I'd like to change [something]
107	I could come out with this solution that
108	I intend to [verb]
109	The easiest solution is probably to [verb]
110	Maybe it can [verb]
111	it might be a good choice to [verb]
112	I would like to [verb] if possible
113	I would create [something] to [verb]
114	It might be a great idea to carry out [something]
115	[someone] could solve [something] by [verb]
116	[something] could be resolved in the way of [verb]
117	A better solution might be [something]
118	We could/should support [something]
119	An efficient way may be [something]
120	Try to create [something] as possible
121	[something] would begin to take affect
122	[something] might be useless
123	The whole idea here could be [something]
#	**Information Seeking**
124	Which [something] may be available?
125	Why would you [verb]?
126	Could [something] be [verb]?
127	would like to know [something]
128	Will [something] be supported?

129	Could [someone] use [something]?
#	**Information Giving**
130	New changes may include [something]
131	Plan could be [something]
132	[something] may use [something]
133	[something] could have been modified
134	[someone] would like to contribute
135	[something] could be removed
136	[something] could support [something]
137	[something] could be added
138	[someone] may update [something]
139	You could always use [something]
140	Probably [something] can help
141	Possibly [something] can be merged
142	[something] might be available
143	[someone] could write code to [verb]
144	You could do [something] by using [something]
145	[someone] declares/announces [something] could [verb]
146	[someone] note that [something] could be [something]
147	[someone] would like to provide [something]
148	There are possibly efforts for [something]
149	[something] might be changed

REFERENCES

[1] G. A. Lewis, T. Mahatham, and L. Wrage. Assumptions management in software development. Technical Report, CMU/SEI-2004-TN-021, 2004.

[2] P. Lago and H. van Vliet. Explicit assumptions enrich architectural models. In: Proceedings of the 27th International Conference on Software Engineering (ICSE), St. Louis, Missouri, USA, pp. 206-214, 2005.

[3] A. Tirumala, T. Crenshaw, and L. Sha. Prevention of failures due to assumptions on software components in real-time systems. ACM SIGBED Review, 2(3): 36-39, 2005.

[4] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch: Why reuse is still so hard. IEEE Software, 12(4): 17-26, 2009.

[5] C. Yang, P. Liang, P. Avgeriou, U. Eliasson, R. Heldal, and P. Pelliccione. Architectural assumptions and their management in industry - An exploratory study. In: Proceedings of the 11th European Conference on Software Architecture (ECSA), pp. 191-207, 2017

[6] A. Steingruebl and G. Peterson. Software assumptions lead to preventable errors. IEEE Security and Privacy, 7(4): 84-87, 2009.

[7] C. Yang, P. Liang, and P. Avgeriou. Assumptions and their management in software development: A systematic mapping study. Information and Software Technology, 94(1): 82-110, 2018.

[8] A. Bonaccorsi and C. Rossi. Why open source can succeed. Research Policy, 32(7): 1243-1258, 2002.

[9] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In: Proceedings of the 6th International Working Conference on Mining Software Repositories (MSR), pp. 137-143, 2006.

[10] S. K. Sowe, I. Stamelos, and L. Angelis. Understanding knowledge sharing activities in free/open source software projects: An empirical study. Journal of Systems and Software, 81(3): 431-446, 2008.

[11] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. van Deursen. Communication in open source software development mailing lists. In: Proceedings of the 10th International Working Conference on Mining Software Repositories (MSR), pp. 277-286, 2013.

[12] C. Bird, A. Gourley, and P. Devanbu. Detecting patch submission and acceptance in OSS projects. In: Proceedings of the 7th International Working Conference on Mining Software Repositories (MSR), pp. 26-29, 2007.

[13] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology, 11(3): 309-346, 2002.

[14] M. M. Lehman. The role and impact of assumptions in software development, maintenance and evolution. In: Proceedings of the 1st International IEEE Workshop on Software Evolvability (IWSE), pp. 3-16, 2005.

[15] ARIANE 5 Flight 501 Failure Report by the Inquiry Board. http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html, 1996.

[16] C. B. Haley, R.C. Laney, J. D. Moffett, and B. Nuseibeh. Using trust assumptions with security requirements. Requirements Engineering. 11(2): 138-151, 2006.

[17] S. K. Sowe, I. Stamelos, and L. Angelis. Understanding knowledge sharing activities in free/open source software projects: An empirical study. Journal of Systems and Software, 81(3): 431-446, 2008.

[18] D. M. Fernandez, W. Bohm, and M. Broy. Artefacts in software engineering: What are they after all? arXiv:1806.00098, 2018.

[19] Y. Ma and S. Fakhoury. Automatic Classification of Software Artifacts in Open-Source Applications. In: Proceedings of the 15th International Conference on Mining Software Repositories (MSR), pp. 398-446, 2018.

[20] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering, 14(2): 131-164, 2009.

[21] C. Yang, P. Liang, and P. Avgeriou. A survey on software architectural assumptions. Journal of Systems and Software, 113 (3): 362-380, 2016.

[22] Oxford English Dictionary. http://www.oxforddictionaries.com/definition/english, accessed on January 15th, 2018.

[23] D.V. Landuyt and W. Joosen. Modularizing early architectural assumptions in scenariobased requirements. In: Proceedings of the 17th International Conference on Fundamental Approaches to Software Engineering (FASE), pp. 170-184, 2014.

[24] H. Ziv, D. Richardson, and R. Klösch. The Uncertainty Principle in Software Engineering, Technical Report, 1997.

[25] A. D. Sorbo, S. Panichella, C. A. Visaggio, M. D. Penta, G. Canfora, and H. C. Gall. DECA: development emails content analyzer. In: Proceedings of the 38th International Conference on Software Engineering (ICSE), Austin, TX, USA, pp. 641-644, 2016.

[26] A. D. Sorbo, S. Panichella, C. A. Visaggio, M. D. Penta, G. Canfora, and H. C. Gall. Development emails content analyzer: Intention mining in developer discussions. In: Proceedings of the 30th International Conference on Automated Software Engineering (ASE), pp. 12-23, 2015.

[27] K. A. de Graaf, P. Liang, A. Tang, W. R. van Hage, and H. van Vliet. An exploratory study on ontology engineering for software architecture documentation. Computers in Industry, 65(7): 1053-1064, 2014.

[28] B. G. Glaser and A. L. Strauss. The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine Publishing, New York, 1967.

[29] S. Adolph, W. Hall, and P. Kruchten. Using grounded theory to study the experience of software development. Empirical Software Engineering, 16(4): 487-513, 2011.

[30] G. Allan. A critique of using grounded theory as a research method. Electronic Journal of Business Research Methods, 2(1): 1-10, 2003.

[31] W. Ding, P. Liang, A. Tang, and H. van Vliet. Understanding the causes of architecture changes using OSS mailing lists. International Journal of Software Engineering and Knowledge Engineering, 25(09&10): 1633-1651, 2015.

[32] C. Yang, P. Liang, P. Avgeriou, U. Eliasson, R. Heldal, P. Pelliccione, and T. Bi. An industrial case study on an architectural assumption documentation framework. Journal of Systems and Software, 134(12): 190-210, 2017.