When and Why Do Software Developers Face Uncertainty?

1st Naoyasu Ubayashi
Kyushu University
Fukuoka, Japan
ubayashi@ait.kyushu-u.ac.jp

2nd Yasutaka Kamei
Kyushu University
Fukuoka, Japan
kamei@ait.kyushu-u.ac.jp

3rd Ryosuke Sato
Kyushu University
Fukuoka, Japan
sato@ait.kyushu-u.ac.jp

***Abstract*—Recently, many developers begin to notice that uncertainty is a crucial problem in software development. Unfortunately, no one knows how often uncertainty appears or what kinds of uncertainty exist in actual projects, because there are no empirical studies on uncertainty. To deal with this problem, we conduct a large-scale empirical study analyzing commit messages and revision histories of 1,444 OSS projects randomly selected from the GitHub repositories. The main findings are as follows: 1) Uncertainty exists in the ratio of 1.44% (average); 2) Uncertain program behavior, uncertain variable/value/name, and uncertain program defects are major kinds of uncertainty; and 3) Sometimes developers tend to take an action for not resolving but escaping or ignoring uncertainty. Uncertainty exists everywhere in a certain percentage and developers cannot ignore the existence of uncertainty.**

***Keywords*-Uncertainty, Empirical Study, OSS Projects**

I. Introduction

Nowadays most developers, regardless of open source software (OSS) or industry projects, admit that uncertainty is a real problem in software development. Indeed, uncertainty has attracted a growing interest among researchers [8], [9], [14], [15], [29], [31], [35], [37]–[40], [48], [49]. Research themes spread over uncertainty of goal modeling, UML modeling, model transformations, and testing. Unfortunately, no one knows how often uncertainty appears or what kinds of uncertainty exist in actual projects, because there are no empirical studies on uncertainty. If developers do not face uncertainty in actual software development, uncertainty might not be an important issue to be tackled in software engineering research.

This paper explores the inside of GitHub [1] in terms of uncertainty to clarify when and why developers face uncertainty in actual projects[2].

Motivation. Table I, several examples of uncertainty in actual software development, shows impressive commit messages in the `GIMP` (GNU Image Manipulation Program) project (January 1, 1997–June 9, 2015). In case of No.2, for example, a committer finally removed the junk functions to resolve uncertainty. As implied in these examples, we assume that uncertainty is a crucial issue in software development. However, state-of-the-art research does not provide an evidence for guaranteeing this assumption. Moreover, it is unclear what kinds of issues exist in actual projects.

Table I
Example of Commit Messages

No.	Keyword	Commit Message in GIMP Project
1.	Unknown	The file builds now and I only see warnings about using unknown Carbon API.
2.	Unknown	Also remove some junk that was there for unknown reasons, this tool has a long history.
3.	Unclear	It is highly unclear when to return FALSE.
4.	Debatable	Whether or not undo memory should be included here is debatable.

1) https://github.com/GNOME/gimp/commit/0351c13a36b01e45be65c616a6d52b28445247af
2) https://github.com/GNOME/gimp/commit/f078a7416c163e743bd19f6f5c0a250a08e8c4c8
3) https://github.com/GNOME/gimp/commit/95c13dad93c2f0f729507507c5aa2c7bb58ca97d
4) https://github.com/GNOME/gimp/commit/35bd3b450d460b8a93bee5544e42870a996fce2e

Contributions. We conduct a large-scale empirical study analyzing commit messages and revision histories of GitHub OSS projects to confirm whether or not our assumption is really true. We address the following research questions:

RQ1: What kinds of uncertainty appear?
Uncertain program behavior (ratio is 36.05 %), uncertain variable / value / name (14.97 %), and uncertain program defects (11.56 %) are major kinds of uncertainty.

RQ2: How do committers deal with uncertainty?
Committers take one of the following actions: considering about uncertainty (47.06 %), resolving uncertainty (38.24 %), and escaping / ignoring uncertainty (14.71 %). In some situations, committers tend to take an action for not resolving but escaping or ignoring uncertainty.

RQ3: When and Why does uncertainty appear?
Uncertainty exists in the ratio of 1.44%. (average). Uncertainty appears shortly after uncertainty-causing commits (median: two changes by two committers). However, there are cases in which the code is repeatedly modified many times from an uncertainty-causing commit. As a reason of uncertainty, the ratio of future requirements is close to half.

Our findings show that developers cannot ignore the existence of uncertainty. The answers to the research questions give us an opportunity for discussing how to support

[1] https://github.com/github

[2] This paper is an extended version of our poster presentation [43].

developers facing uncertainty. It is effective to provide the uncertainty-aware software development environment reflecting our findings.

Paper Organization. The remainder of this paper is organized as follows. We survey the definition of uncertainty in Section II. Section III describes the design of our study, while Section IV presents the results. Section V discloses the threats to the validity. Section VI shows the related work. Section VII draws conclusions.

II. Classification of Uncertainty

Uncertainty is an abstract concept. Although many people might feel that *definition of uncertainty is uncertain*, there is a consensus of its definition in the research community.

A. Two Types of Uncertainty

First, we show the most representative definition of uncertainty. In general, there are two types of uncertainty affecting software development: *Known Unknowns* and *Unknown Unknowns* [8]. In the *Known Unknowns*-type, there are uncertain issues in the process of software development. However, these issues are known and shared among the stakeholders including developers and customers. For example, there are alternative requirements although it is uncertain which alternative should be selected. On the other hand, in the *Unknown Unknowns*-type, it is uncertain what is uncertain. This type is difficult to be dealt with, because it is unpredictable what kind of issues will appear in the future. Due to this reason, current state-of-the-art research mainly focuses on *Known Unknowns*-type uncertainty.

B. Taxonomic Classification

Perez-Palacin, D. and Mirandola, R. [33] provide a systematic review on uncertainty and summarize as follows: *The most used definitions of uncertainty simply distinguish between natural variability of physical processes (i.e., aleatory or stochastic uncertainty) and the uncertainties in knowledge of these processes (i.e., epistemic or state-of-knowledge uncertainty).*

They propose the three-dimensional classification model consisting of *Location*, *Level*, and *Nature* by referring Walker et al.'s work [45]. The main purpose of this model is to classify uncertainties appearing in self-adaptive systems.

Location: Location is categorized into *Context*, *Structural*, and *Input parameters*. Context uncertainty is an identification of the boundaries of a model (or design). Structural uncertainty is contained in a system model itself. The last uncertainty is caused by the vague input parameter values from the real world.

Level: Level is categorized into four orders. In the first order of uncertainty, the subject lacks knowledge about something but a developer is aware of such lack (i.e., *Known Unknowns*). The second order indicates lack of knowledge and lack of awareness (i.e., *Unknown Unknowns*). The third order indicates lack of process to find out the lack of awareness. The fourth order indicates uncertainty about orders of uncertainty (i.e., meta-level uncertainty).

Nature: Nature is categorized into *Aleatory* and *Epistemic*.

Example. Although the main target of the three-dimensional classification model is self-adaptive systems, the model is well-formed and basically applicable to other application domains. For example, uncertain requirements from a business customer can be classified as follows: 1) the location is *Context* (uncertainty exists on the boundary between a system and the customer's world, because this boundary is changeable due to the customer's changeable requirements to the system); 2) the level is the first order (a developer knows that system requirements will be changed but does not know what changes are made by the customer); and 3) the nature is epistemic (the future decisions of the customer are unknown).

This paper uses the three-dimensional classification model to analyze uncertainties appearing in GitHub OSS projects.

III. Empirical Study Setup

In this section, we explain data set and an approach to identifying uncertainties.

A. Data Set

The GitHub repositories as of February 2017 are used in this empirical study. 1,444 projects are randomly selected from GitHub and used as the data set. Repositories in GitHub are not always software projects [25]. For example, there are repositories for storing only configuration files. Hence, we filter away meaningless repositories from our data set. Our data set contains various project data ranging from large ones (e.g. Linux) to small ones. We use commit messages in our investigation, because text data included in commit messages can be a clue to analyze the tendency of uncertainty. Meaningless commit messages such as automatically generated ones are excluded from the target of analysis.

B. Identifying Uncertainty

In this study, we assume that a committer deals with some kinds of uncertain concerns when he or she modifies the code and commits with the message containing the keywords related to uncertainty. We call these commits and keywords *uncertainty commits* and *uncertainty keywords*, respectively. Of course, a commit message containing uncertainty keywords does not always indicate an actual uncertainty commit. On the other hand, uncertain concerns might be dealt with even when a committer writes a commit message that does not contain uncertainty keywords. In general, it is not easy to identify uncertainty automatically and correctly, because we have to understand the deep semantics of the target commits that can be affected by not only code modifications but also requirements or design documents. In some cases, these documents may not be

Table II
UNCERTAINTY KEYWORDS

Synonyms for "Uncertainty" (Oxford American Writer's Thesaurus)
debatable, undetermined, unsure, unpredictable, unforeseeable, incalculable, risky, chancy, dicey, informal, iffy, vague, ambiguous, unknown, unascertainable, obscure, arcane, changeable, irregular, unreliable, unsettled, erratic, fluctuating, doubtful, dubious, undecided, irresolute, vacillating, unclear, ambivalent, hesitant, tentative, faltering, unconfident, may, might, probably, fuzzy

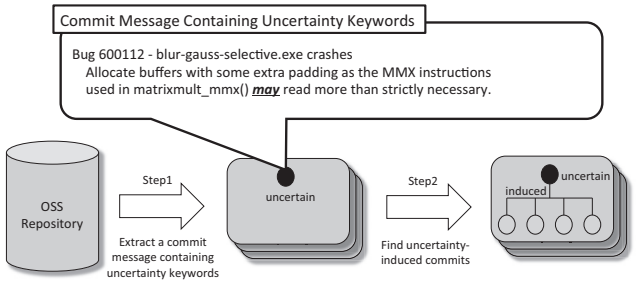

Figure 1. Modified SZZ Algorithm for Uncertainty

stored in project repositories. Taking these situations into account, it is practically reasonable to admit our assumption as a first step approximation of uncertainty identification. In our study, not only automated analysis based on text mining but also manual inspection is performed to avoid mis-judging of uncertainty commits.

Keyword List for Uncertainty. The uncertainty keywords shown in Table II are used as a preliminary dictionary for determining whether a commit deals with uncertainty or not. These keywords, synonyms for "Uncertainty", are extracted from Oxford American Writer's Thesaurus [28].

Historical Uncertainty Identification. An uncertainty commit can be affected by a series of previous code modifications. We need to detect not only an uncertainty commit itself but also commits affecting the uncertainty commit in order to answer RQ3. We adopt the SZZ algorithm [41] to automatically identify the changes that eventually caused uncertainty commits as illustrated in Figure 1. We slightly changed the original SZZ algorithm, because it does not find uncertainty-causing changes but bug-introducing changes. The original algorithm links each bug fix to the program code change introducing the original bug by combining information from the version archive such as Git with the issue tracking system such as Bugzilla. Our algorithm consists of two steps. First, it identifies the change related to uncertainty. Our algorithm searches uncertainty keywords shown in Table II. The second step identifies when the uncertainty is potentially introduced or when the code snippets eventually affecting the uncertainty commit are induced. We use the diff command to locate the lines that were changed by the uncertainty commit. Then, we use the annotate command to trace back to the last revision that changed those patched lines.

C. SE Dictionary for Uncertainty

Unfortunately, the preliminary dictionary consisting of uncertainty keywords shown in Table II is not always suitable for analyzing uncertainty in software development, because these keywords are not only general but also ambiguous in many situations. It is necessary to construct a dictionary specific to SE (Software Engineering).

To deal with this challenge, we perform association rule mining to find what kinds of words appear with accompanying uncertainty keywords in uncertainty commits. By picking up the words related to SE, we can observe what kinds of uncertainty appear in actual software development. We focus on program descriptions, program defects, and committer's actions. The program description words are related to program behavior (e.g. *Call*), program data (e.g. *Variable*), or program implementation (e.g. *Function*). By focusing these words, we can observe in which program portion uncertainty appears. If the word *Call* appears frequently with uncertainly keywords, we understand that uncertainty tends to appear when designing or implementing method or function calls. The program defect words are related to *Bug*, *Error*, or *Problem*. If these words appear frequently with uncertainty keywords, we can observe that uncertainty is likely to be related to program defects. Committer's actions are important to analyze how they behave when facing uncertainty. We regard a set of pairs `(Uncertainty keyword, Program description word)`, `(Uncertainty keyword, Program defect)`, and `(Uncertainty keyword, Committer's Action)` as an SE dictionary for uncertainty.

We used a text analysis method based on tf-idf (Term Frequency-Inverse Document Frequency), a numerical statistic intended to reflect how important a word is to a document (a set of uncertainty commit messages in our study). Term frequency in our case is the number of occurrence times of a word in uncertainty commit messages. Inverse document frequency is a statistical interpretation of term specificity.

Our approach is divided into the following three steps.

- Step 1: Important words associated with uncertainty keywords are extracted from a set of uncertainty commit messages by calculating tf-idf.
- Step 2: Only the nouns and verbs are selected from the result of Step 1, because nouns and verbs can capture the characteristics of program code and development activities. By extracting nouns and verbs occurring with uncertainty keywords, we can analyze what kinds of program elements are related to uncertainty and what kinds of activities are linked to uncertainty.
- Step 3: Nouns and verbs extracted in Step 2 are categorized to analyze the uncertainty tendency observed in OSS projects.

Details of each step are described below.

Step 1: Extracting Words Associated with Uncertainty Keywords: In general, the reasons and procedures of code modifications are described in a commit message. Commit messages are written in natural languages and can be dealt with using natural language processing (NLP) techniques. Firstly, we extract characteristic words associated with each uncertainty keyword from a set of uncertainty commit messages included in each OSS project by calculating tf-idf values. Secondly, characteristic words of all 1,444 projects are obtained by merging the characteristic words extracted in each project. Thirdly, we compare the tf-idf value of each characteristic word obtained from only the uncertainty commits with that of the same characteristic word obtained from all commits. Lastly, we extract only the characteristic words having the tf-idf value whose difference is zero or more. The larger the difference of the tf-idf value of a characteristic word is, the more the word is strongly associated with uncertainty.

Step 2: Extracting Only Nouns and Verbs: As shown in Table II, uncertainty keywords are mainly adjectives or adverbs. We extract only nouns and verbs to understand what is performed in a commit. Verbs show the activities associated with uncertainty represented by adjectives or adverbs. Nouns shows the targets of these activities. In Step 2, we use the openNLP package[3] included in R, an open source programming language and software environment for statistical computing. Using the openNLP package, we can add tags to words in sentences using the Maxent model [34] that is developed based on The Penn Treebank [30], an English Part of Speech (POS) tag database. In our case, tags indicating noun or verb are attached to characteristic words extracted in Step 1. Table III is an example of uncertainty-specific characteristic word list that picks up the words associated with the uncertainty keyword *Unknown*. The mean is an average difference of the tf-idf value explained in the above. This list is sorted in descending order of the mean value. The higher the rank is, the more the word appears with *Unknown*. The result of Table III shows that *Unknown Type*, *Unknown Error*, and *Unknown Message* often appear in GitHub commit messages. This result is close to our expectations, because we often encounter these kinds of descriptions in software development. Each mean value in Table III is small, because the words such as *Type*, *Error*, and *Message* are ordinary terms and commonly used in commit messages. However, each mean value indicates that the rate of combinational usages with *Unknown* is slightly higher than ordinary word usages. We consider that subtle difference is important to explore uncertainty in software development, because uncertainty is basically an exceptional situation comparing other development activities. CV (Coefficient of Variation) [3] is a standardized measure of dispersion of a probability distribution or frequency distribution. A

[3]https://opennlp.apache.org

Table III
WORDS ASSOCIATED WITH *Unknown*

Word	Mean	CV	#Projects
Type	0.01874571	0.853236928	808
Error	0.018047575	0.913466452	903
Message	0.016683953	0.901754048	590
Ignore	0.016163827	1.080568182	528
Return	0.015338545	0.8766055	677
Handle	0.014811577	1.105870303	605
Reason	0.012827254	0.873758822	666
Name	0.010824656	0.98421489	531
Report	0.009979287	1.080975554	539
Case	0.009530822	1.082011154	500
Cause	0.007997205	1.138618122	538
Will	0.00708426	0.902794009	592
Warn	0.016426316	0.911621722	479
Fail	0.01033176	1.012432553	543
Trigger	0.008507029	1.229532611	507

characteristic word spreads over multiple projects if its CV value is large. That is, the word is not specific to a certain project but common to many projects. The forth column shows the number of projects in which the corresponding word is used. For example, *Unknown Type* appears in 808 projects out of the all 1,444 projects. Usage rate is 0.60 (= 808/1444). This indicates that *Unknown Type* is one of the popular usages in uncertainty.

Step 3: Classifying Uncertainty-Specific Characteristic Words: In this step, uncertainty-specific characteristic words related to software development are picked up and categorized into three groups: program descriptions, program defects, and committer's actions as shown in Table IV. Other general words such as *Like*, *See*, *Take* are excluded from this analysis, because these words are not specific to software development.

Constructed Dictionary. Our SE dictionary for uncertainty is shown in Table V (dictionary for uncertain program descriptions and defects consisting of 147 elements) and Table VI (dictionary for committer's actions consisting of 34 elements). This dictionary contains top-20 uncertainty keywords frequently appearing in GitHub commit messages and reflects the tendency in actual software development. For example, `(Ambiguous, Value)`, `(Ambiguous, Type)`, and `(Ambiguous, Avoid)` are dictionary elements.

Although Step 1 and 2 were performed automatically by writing scripts, Step 3 was manually executed by assigning three master students and one faculty member as evaluators.

IV. STUDY RESULTS

In this section, we present the results of our study. For each research question, we discuss its motivation, explain our approach to addressing it, and present our observations.

RQ1: What Kinds of Uncertainty Appear?

Motivation. Most people might consider that developers suffer from many kinds of uncertainty: program defects

Table IV
CLASSIFICATION OF UNCERTAINTY-SPECIFIC CHARACTERISTIC WORDS

Category	Explanation
Program Descriptions	Words related to program behavior, program data, or program implementation.
Program Defects	Words related to bugs, errors, warnings, or problems.
Committer's Actions	Words related to actions performed by committers to deal with uncertainty.

Table V
SE DICTIONARY FOR UNCERTAINTY: PROGRAM DESCRIPTIONS AND DEFECTS (147 ELEMENTS)

No.	Uncertainty Word	Program Descriptions	Program Defects
1	Ambiguous	Call, Case, Differ, Name, One, Value, Type	Error, Warn
2	Arcane	Build, Column, Damage, Empty, Entry, Shot, Stack, Trigger	Error, Problem
3	Changeable	Cache, Default, Disk, Field, Offset, Pass, Set	Error
4	Debatable	Case, Get, Name, Place, Put, Seem, System	
5	Dubious	Bit, Case, Code, Get, Value, Seem, Use	Error, Warn
6	Doubtful	Look, One, Work	
7	Erratic	Affect, Behavior, Cause, Logic, Part, Result	Bug, Issue
8	Fuzzy	Code, Get, Input, Number, Option, Require, Test, Way	
9	Irregular	Approach, Break, Condition, Part, Place, Pass, Switch, Usage, Value, Work	
10	May	Case, Cause, Contain, Differ, Happen, Mean	
11	Might	Case, Cause, Differ, Get, Happen, Look, Trigger	
12	Obscure	Call, Case, Cause, Code, Get, Result	Bug, Error, Problem
13	Probably	Cause, Get, Seem, Way, Work	
14	Risky	Case, Data, Run, Security	
15	Tentative	Call, Definition, Method, Way, Work	Bug, Problem
16	Unclear	Differ, Name, Reason, User, Value, Seem	
17	Unknown	Case, Cause, Fail, Message, Name, Reason, Report, Return, Trigger, Type	Error, Warn
18	Unreliable	Case, Detect, Result, Run, Set, Test, Trigger	
19	Unsure	Case, Result, Seem, Work	
20	Vague	Attempt, Cause, Document, Function, Get, Line, Return, Set	Error

Table VI
SE DICTIONARY FOR UNCERTAINTY: COMMITTER'S ACTIONS (TOTAL 34 ELEMENTS)

No.	Uncertainty Word	Committer's Actions
1	Ambiguous	Avoid, Rename
2	Arcane	
3	Changeable	Check
4	Debatable	Think, Want
5	Dubious	Avoid, Remove, Replace
6	Doubtful	See, Want
7	Erratic	Handle, Workaround
8	Fuzzy	
9	Irregular	
10	May	Need, Want
11	Might	Need, Want
12	Obscure	Need
13	Probably	Need, Want
14	Risky	Avoid, Need, Reduce
15	Tentative	Allow, Implement, Support
16	Unclear	
17	Unknown	Handle, Ignore
18	Unreliable	Change, Check, Need, Remove
19	Unsure	Need, Think
20	Vague	Think

whose reason is uncertain, strange program behavior, unexpected value of a program variable, and program code whose implementation policy is doubtful. Unfortunately, these subjective observations are merely guesses, because no evidence is provided from state-of-the-art research. Even if this assumption is correct, it is still unclear what kinds of uncertainty appear in actual software development. We provide an evidence showing that our assumption is not wrong.

Approach. We approximate the real world tendency by using the dictionary shown in Table V and derive an answer to RQ1. Table VII summarizes the frequency count of the words associated with uncertainty keywords. If there are dictionary elements `(Unknown, Type)` and `(Vague, Type)`, the count number of the word `Type` is two. By observing the occurring number of the words classified into program descriptions and defects in Table VII, we can understand what kinds of uncertainty tend to appear in actual software development, because our dictionary is constructed from real OSS repositories.

Results. The ratio of each uncertainty type is as follows: uncertain program behavior 36.05 % (= 53/147), uncertain variable / value / name 14.97 % (= 22/147), uncertain program defects 11.56 % (= 17/147), uncertain cases 8.16 % (= 12/147), uncertain program implementation 7.48 % (= 11/147), uncertain cause 6.12 % (= 9/147), uncertain artifacts & activities 4.79 % (= 7/147), uncertain non-functional requirements 0.68 % (= 1/147), and other 10.20 % (= 15/147). We observe each kind of uncertainty in details.

- *Uncertain program behavior:* The words associated with program behavior appear most frequently. These words include *Behavior*, *Call*, *Differ*, *Fail*, *Get*, *Return*, *Trigger*, and *Work*. Uncertainty tends to appear when we cannot obtain expected program behavior.
- *Uncertain variables, values and names:* The words associated with program data also appear frequently. These words include *Field*, *Name*, *Type*, and *Value*.

Table VII
FREQUENCY OF UNCERTAINTY-SPECIFIC CHARACTERISTIC WORDS

Category	Sub Category	Words	Count	Ratio [%]
Program Descriptions	1. Uncertain Program Behavior	Affect, Behavior, Break, Call, Condition, Damage, Detect, Differ, Fail, Get, Happen, Message, Logic, Pass, Put, Return, Result, Run, Set, Shot, Switch, Trigger, Usage, Use, User, Work	53	36.05
	2. Uncertain Variable, Value, and Name	Cache, Column, Data, Disk, Empty, Field, Input, Name, Number, Offset, Place, Stack, Type, Value	22	14.97
	3. Uncertain Cases	Case, Default, Option	12	8.16
	4. Uncertain Program Implementation	Code, Entry, Definition, Function, Line, Method, System	11	7.48
	5. Uncertain Cause	Cause, Reason	9	6.12
	6. Uncertain Artifacts & Activities	Approach, Attempt, Build, Document, Test, Report	7	4.79
	7. Uncertain Non-functional Requirements	Security	1	0.68
	8. Other	Contain, Look, Mean, One, Require, Seem, Way	15	10.20
Program Defects	Uncertain Program Defects	Error, Bug, Issue, Problem, Warn	17	11.56
Total			147	100

Top-11 Words: Case (10), Cause (7), Error (7), Get (7), Seem (5), Work (5), Differ (4), Name (4), Result (4), Trigger (4), Value (4)

Uncertainty tends to appear when defining or using variables, types, or values.

- *Uncertain cases:* *Case*, *Default*, and *Option* appear in the situations such as alternative / optional design choices and unexpected behavioral cases. In the former case, uncertainty tends to occur in program design and implementation. For example, developers often worry which algorithm should be selected to improve performance. In the latter case, two words *Case* and *Cause* are listed up simultaneously in *May*, *Might*, *Obscure*, and *Unknown*.
- *Uncertain program implementation:* The words associated with program implementation appear at a certain rate. These words includes *Code*, *Function*, and *Method*. The expressions *Dubious Code*, *Fuzzy Code*, *Obscure Code*, and *Vague Function* indicate that a committer feels when and why the target program code was implemented. In general, most major OSS projects have a long history and many developers write and push the code. In such a case, the context or background of program implementation may be forgotten. As the another case, the expressions such as *Tentative Method* indicates that a committer might push the temporally implemented code to a repository. This kind of action may raise another uncertainty for other developers.
- *Uncertain cause:* The words *Cause* or *Reason* are associated with program behavior. Although this category can be merged into the *uncertain program behavior*, we consider that the cause of uncertain behavior is preferred to be distinguished. The word *Cause* is associated with *Erratic*, *May*, *Might*, *Obscure*, *Probably*, *Unknown*, and *Vague*. The word *Cause* is used when an unexpected case occurs but the reason is unknown. Figure 2 [4] is an example.

[4]https://github.com/alrieckert/freepascal/commit/6a7f8d70da3af5c202db55189d3184d92861c6f8

Figure 2. Example of *Unknown* + *Reason*

- *Uncertain artifacts & activities:* The words *Document* and *Test* are associated with uncertain artifacts. The expressions *Vague Document* or *Unreliable Test* appear in commit messages. The words such as *Approach* and *Attempt* are associated with uncertain activities.
- *Uncertain non-functional requirements:* The uncertainty keywords such as *Risky* and *Unreliable* are related to non-functional requirements. The word *Security* appears with the uncertainty-keyword *Risky*. Security is one of the important non-functional requirements. It is not always easy to know how much secure a program should be.
- *Uncertain Program Defects:* The words associated with program defects appear frequently. The words include *Bug*, *Error*, *Problem*, and *Warn*. We consider that a certain amount of defects are related to uncertainty.

Answer to RQ1
The following kinds of uncertainty tend to appear: uncertain program behavior 36.05 %, uncertain variable / value / name 14.97 %, uncertain program defects 11.56 %, uncertain cases 8.16 %, uncertain program implementation 7.48 %, uncertain cause 6.12 %, uncertain artifacts & activities 4.79, uncertain non-functional requirements 0.68 %, and other 10.20 %.

RQ2: How Do Committers Deal with Uncertainty?

Motivation. We could understand what kinds of uncertainty appear in GitHub projects. However, it is not yet clear what kinds of actions committers take when facing uncertainties.

Table VIII
FREQUENCY OF WORDS RELATED TO COMMITTER'S ACTIONS

Category	Sub Category	Words	Count	Ratio [%]
Committer's Actions	1. Considering about Uncertainty	See, Need, Want, Think	16	47.06
	2. Resolving Uncertainty	Allow, Change, Check, Handle, Implement, Reduce, Remove, Rename, Replace, Support	13	38.24
	3. Escaping or Ignoring Uncertainty	Avoid, Ignore, Workaround	5	14.71
Total			34	100

Top-3 Words: Need (7), Want (5), Think (3)

Approach. We use the dictionary shown in Table VI to answer RQ2. Table VIII summarizes the frequency count of the words associated with uncertainty keywords. For example, the count number of the word `Remove` is two, because there are dictionary elements `(Dubious, Remove)` and `(Unreliable, Remove)` in Table VI. We can observe the tendency of committer's actions when facing uncertainties by analyzing Table VIII.

Results. The ratio of each committer's action is as follows: considering about uncertainty 47.06 % (= 16/34), resolving uncertainty 38.24 % (= 13/34), and escaping or ignoring uncertainty 14.71 % (= 5/34). We observe each kind of action in details.

- *Considering about Uncertainty:* The words associated with activities for considering about uncertainty appear frequently. The words include *Need*, *Think*, and *Want*. However, these words do not directly indicate activities for resolving, escaping, or ignoring uncertainty.
- *Resolving Uncertainty:* The words such as *Change*, *Handle*, *Reduce*, *Remove*, *Rename* and *Replace* are associated with activities for resolving uncertainty.
- *Escaping or Ignoring Uncertainty:* The words associated with activities for escaping or ignoring uncertainty appear sometimes. The words include *Avoid*, *Ignore*, and *Workaround*. In this case, the problem might not be essentially resolved. Our result indicates that this is not a rare case.

When a committer writes a commit message, he or she resolves uncertainties, makes some kinds of code modifications, and pushes a new revision to the repository. However, there are possibilities that a committer takes no action and does not write any commit messages even if he or she faces uncertainty. Unfortunately, our study cannot catch such a case. This is a limitation of our study.

Answer to RQ2
The following kinds of committer's actions tend to appear: considering about uncertainty 47.06 %, resolving uncertainty 38.24 %, and escaping / ignoring uncertainty 14.71 %. Committers write a commit message to resolve uncertainty. In some situations, however, committers tend to take an action for not resolving but escaping or ignoring uncertainty.

RQ3: When and Why Does Uncertainty Appear?

Motivation. In RQ1 and RQ2, we analyzed uncertainty by inspecting commit messages. However, as another aspect, we should analyze uncertainty from the historical and causal viewpoints. In RQ3, we address the following three sub research questions: RQ3-1) How often do uncertainty commits appear?; RQ3-2) How many time are code snippets modified until an uncertainty commit eventually appear?; and RQ3-3) Why are uncertainty commits raised? It is important to answer the reasons why uncertainty appears. If we can know the reasons, we are able to provide tool support for dealing with uncertainty.

Approach. As explained in Section III, we adopt the modified SZZ algorithm to automatically identify the changes that cause uncertainty commits. In this paper, we randomly selected ten projects out of GitHub 1,444 projects. We reduced the number of projects, because the modified SZZ algorithm needs large amount of computational resource to trace back to a series of revision histories. The left side of Table IX shows the data set used in this analysis. Each selected project contains 500–5,000 commits, because a sufficient number of commits is needed for the validity of analysis.

To answer RQ3-3, we analyze the text of commit messages manually, because it is difficult to understand the real reasons by only using the automated text analysis approach. We select the `ead` project in Table IX as an example, because the ratio of code changes (median) is highest and we expect that an interesting tendency can be observed (The answer to RQ3-2 is shown below). We qualitatively analyze the characteristics of uncertainty commits using the three-dimensional classification, a model consisting of *Location*, *Level*, and *Nature* shown in Section II. By reading the commit messages manually, we checked the reason of uncertainty, the location of uncertainty (*Structural* or *Context*), uncertainty level (*Known Unknowns* or not), and the nature of uncertainty (*Epistemic* and *Aleatory*).

Results.

Answer to RQ3-1 (Occurrences of Uncertainty). The average ratio of uncertain commits is 1.44% (max 3.77% and min 0.26%) as shown in the middle columns of Table IX. This value is not high. However, all of the OSS projects contain uncertainty commits. In case of the `piglit` project, there are 129 uncertainty commits. The result of RQ3-1 shows that uncertainty exists everywhere in a certain

Table IX
UNCERTAINTY FROM HISTORICAL VIEWPOINT

No.	Project	Period	Main Language	All Commits	Uncertainty Commits	Ratio [%]	Code Changes			Committers		
							Max	Min	Median	Max	Min	Median
1	cadcVOFS	2009/07/24 - 2015/12/03	Python	3,007	20	0.67	18	1	2	8	1	2
2	cdec	2010/06/22 - 2014/10/25	C++	2,183	16	0.73	17	1	2	5	1	2
3	class2go	2012/06/04 - 2013/10/09	Python	4,595	23	0.50	8	1	2	3	1	1
4	ead	2013/10/28 - 2015/10/10	Java	4,611	12	0.26	12	1	2.5	4	1	2
5	fail2ban	2004/10/07 - 2016/04/23	Python	3,412	51	1.49	35	1	2	9	1	2
6	KeeFox	2008/12/24 - 2016/01/23	JavaScript	743	28	3.77	160	1	2	6	1	1
7	myblendercontrib	2010/04/12 - 2016/01/26	Python	2,432	56	2.30	13	1	2	5	1	2
8	myria	2012/08/25 - 2016/02/12	Java	4,111	89	2.16	27	1	2	7	1	2
9	picard	2009/05/14 - 2014/06/10	Java	1,511	21	1.39	10	1	1	6	1	2
10	piglit	2007/05/24 - 2013/08/14	C	4,403	129	2.93	8	1	1	6	1	2

Table X
QUALITATIVE ANALYSIS OF UNCERTAINTY COMMITS (THE `ead` PROJECT)

No.	HashID	Keyword	Reason of Uncertainty	Location	Level	Nature
1	753fc1dbc7ef2c3cfe7754685ff405fcd0509b6c	May	Future Requirements	Context	1 (Known-Unknowns)	Epistemic
2	7b92e50f75f2402c7e69c231133aaf103c68457a	Probably	Refactoring	Structural	1 (Known-Unknowns)	Epistemic
3	7e5495a9b81976e1d55ffe1ade435ae404339524	May	Preventive Maintenance	Context	1 (Known-Unknowns)	Aleatory
4	83f78b7917bb6486df2b7ae0ddc0dfd10edda7aa	May	Future Requirements	Context	1 (Known-Unknowns)	Epistemic
5	890c8558c2ccb8631c4c2543e87b90e94696f5c7	Unpredictable	Preventive Maintenance	Context	1 (Known-Unknowns)	Aleatory
6	8a4cc314cd632b198d5677ba208ebc5adccdb3b2	Probably	Refactoring	Structural	1 (Known-Unknowns)	Epistemic
7	af35b3642ebccc6a2b84953efc4b7cb5357bb48a	May	Refactoring	Structural	1 (Known-Unknowns)	Epistemic
8	c4dbb3dec35fbfece429d59d87a6b6c5a5802457	May	Future Requirements	Context	1 (Known-Unknowns)	Epistemic
9	f37bcde398c46c93a7d4d3c07efcbb22fe23b1af	May	Future Requirements	Context	1 (Known-Unknowns)	Epistemic

percentage and developers cannot ignore the existence of uncertainty.

Answer to RQ3-2 (Change Frequency of Code Snippets). Someone might consider that developers tend to go back and forth by modifying the same code region when they face uncertainty, because they worry about how to write the correct code (in case of the unknown cause of bugs) or the good code (in case of refactoring). However, the median of the code modification times is two as shown in the right side of Table IX. Contrary to our expectation, uncertainty commits appear shortly after uncertainty-causing commits. However, there are cases in which the code is repeatedly modified many times. For example, the maximum count of code changes is 160 in case of the `KeeFox` project.

Answer to RQ3-3 (Reasons of Uncertainty). Table X shows the results. Although there are twelve uncertainty commits in the `ead` project, we exclude three commits that do not actually contain uncertainty. The reasons of uncertainty are categorized into *Future Requirements* 44.44 % (= 4/9), *Refactoring* 33.33 % (= 3/9), and *Preventive Maintenance* 22.22 % (= 2/9). In the `ead` project, uncertainty tends to appear in the following situations: 1) uncertain whether or not this feature will be needed in the future (*Future Requirements*); 2) uncertain whether or not this code should be refactored (*Refactoring*); and 3) uncertain whether or not this code may cause problems in the unknown context (*Preventive Maintenance*).

Although these results depend on the `ead` project, they raise a thoughtful discussion as follows. The ratio of *Future Requirements* is close to half. This result indicates that requirements management and change management are important in uncertainty-aware software development. *Location* and *Nature* of the uncertainty commits categorized into *Future Requirements* are *Context* and *Epistemic*, because future requirements are affected by stakeholders. *Refactoring* for uncertainty is also important in tool support. We have to provide the tool features that can change the code snippets between *Certain* and *Uncertain*. *Location* and *Nature* of the uncertainty commits categorized into *Refactoring* are *Structural* and *Epistemic*, because refactoring is to improve the program structure. *Preventive Maintenance* mainly originates in unexpected program behavior. *Location* and *Nature* of the uncertainty commits categorized into *Preventive Maintenance* are *Context* and *Aleatory*. The characteristics of *Preventive Maintenance* are different from those of *Future Requirements* and *Refactoring*. In terms of tool support, probabilistic testing or verification are useful in performing *Preventive Maintenance*, because uncertainty location is context that is not always controlled from a program and a developer has to deal with stochastic uncertainty. Although the ratio of *Preventive Maintenance* is relative low in the `ead` project, we consider that this ratio becomes high in case of system software products such as operating systems and IoT (Internet of Things) systems. However, we predict that the ratio of *Preventive Maintenance* in ordinary products such as utility software is similar to the value in the *ead* project.

Limitations. Although all of the uncertainties in Table X are categorized into *Known Unknowns*, this observation is not always generalized at this point. In principle, *Unknown*

Unknowns cannot be caught by our current approach. When uncertain concerns are *Unknown Unknowns*, a developer may not conduct a commit for them because he or she is unaware of the unknown concerns. In such a case, we may not find an *Unknown Unknowns* type of uncertainty from only the commit messages.

Answer to RQ3
Uncertainty exists in the ratio of 1.44% (average) and developers cannot ignore its existence (RQ3-1). Uncertainty commits appear shortly after uncertainty-causing commits (median: two changes by two committers). However, there are cases in which the code is repeatedly modified many times (RQ3-2). As a reason of uncertainty, the ratio of future requirements is close to half (RQ3-3).

V. Threats to Validity

In this section, we discuss on threats to validity in terms of external, construct, and internal validity.

External Validity. External validity indicates the extent to which the results of our study can be generalized. In the study for RQ1 and RQ2, we used the commit messages from 1,444 GitHub OSS projects. We consider that the number of projects is sufficient to guarantee the validity of the obtained results. On the other hand, in RQ3, only the ten projects were used due to the limitation of the computational resource needed for using the modified SZZ algorithm. However, we consider ten is a sufficient number to guarantee the validity of our analysis.

In RQ3-3, we took an approach of case study. We consider this approach is appropriate, because we have to deeply check the contents of each commit message to know why uncertainty appears. Of course, the scale of manual analysis is insufficient and we should increase the number of the case studies. For this purpose, it is necessary to improve our analysis procedure (i.e., manual inspection) by adopting other techniques such as machine learning. This is our future work. However, it is essentially difficult to determine whether or not a commit is really an uncertainty commit, because there are cases in which a commit not containing uncertainty keywords might be an uncertainty commit. In these cases, we have to check the contents of these commit messages semantically. Although this problem exists in RQ1 and RQ2, these research questions mainly focus on finding the co-occurrence of uncertainty keywords and associated SE terms. At this point, we cannot determine whether or not machine learning is really useful for this purpose. Nevertheless, we admit that further improvement is needed.

Construct Validity. Construct validity indicates the appropriateness of inferences made on our measurements. Our study was conducted only for the commit messages. This approach has the following issues: 1) actual code is not essentially taken into account although historical code modifications are analyzed in RQ3; 2) we cannot take into account uncertainty that cannot be caught by commit messages; and 3) the quality of our analysis depends on the identified keywords and semantics of the messages.

For the first issue, we plan to trace the relation between commit messages and code snippets by searching the code regions from the method names or variable names appearing in the commit messages. The second issue is still difficult to be resolved. For example, *Unknown Unknowns* are not always caught by only analyzing commit messages as mentioned in the answer to RQ3. As one possibility, text included in requirements, design documents, mailing lists, and issue tracking systems might be able to be used for this purpose. For the third issue, we consider that the quality can be improved with techniques from the natural language processing community, especially some techniques can be used to catch the semantics of the short sentences in commit log. This can contribute to reduce false positives in identifying uncertainties.

Internal Validity. Internal validity indicates the extent to which a causal conclusion based on our study is warranted. In our study, one commit message can be categorized into multiple categories because of having multiple reasons of the uncertainty or multiple handling of uncertainty. This situation may occur. However, we tentatively categorize such commit into the most preferable group, because we want to calculate the uncertainty-commits ratio. A technique similar to an approach dealing with tangled-commits may be introduced to resolve this problem. This is our future work.

Some of the uncertainties might not have been caused by the program code or structure but by the experience of a developer. For instance, a developer that has just joined the project might have more uncertain commits due to their lack of understanding of the project. We have to take into account this possibility. We consider that a part of these uncertainties can be caught by analyzing texts contained in commit messages, informal documents, or mailing lists. This issue should be resolved in the future.

Recent study by Devanbu et al. [5] shows that there can be difference between the beliefs of practitioners and actual data and a strong interplay of belief and evidence in software practice is needed. Uncertainty is a real problem in industry. The first author of this paper worked in industry as a software engineer for twenty years before moving to academia. He suffered from many kinds of uncertainties: it was uncertain when user requirements changed; functional specifications could not be finalized at the initial requirements elicitation phase; and there were multiple design choices due to non-functional requirements such as performance. This paper is motivated by our experience in industry.

Although our study has some limitations as mentioned above, the obtained results are basically fitted to our experiences.

VI. Related Work and Discussion

In this section, we situate this paper with respect to the state-of-the-art research on uncertainty. After that, we discuss how to make use of our research results to improve our software development.

A. State-of-the-art Research

Most of the state-of-the-art studies focus on *Known Unknowns*. As a representative work, a method for expressing *Known Unknowns* using a partial model is proposed in [14], [16]. A partial model is a single model containing all possible alternative designs of a system and is encoded in propositional logic. We can check whether or not a model including uncertainty satisfies some interesting properties. Research themes spread over uncertainty of requirements modeling, software architecture, model transformations, programming, testing, verification, and performance engineering. In [39], a partial model is applied to uncertainty in requirements to address the problem of specifying uncertainty within a requirements model, refining a model as uncertainty reduces, providing meaning to traceability relations between models containing uncertainty, and propagating uncertainty-reducing changes between related models. In [1], [11], [13], [26], uncertainty is explored in terms of software architecture. Letier, E. et al. present a support method for evaluating uncertainty, its impact on risk, and the value of reducing uncertainty in requirements and architecture [27]. In [40], a method for change propagation in the context of model uncertainty is proposed. Most of these studies focus on epistemic uncertainty. In [17], uncertainty is explored in terms of software product line. Uncertain$< T >$, a simple probabilistic programming language for letting programmers without statistics expertise easily and correctly compute with estimates [2]. Uncertain$< T >$ deals with aleatory uncertainty. Modularity for uncertainty, in which *Known Unknowns* can be expressed as a first-class software module, is proposed in [19], [20], [46]. Elbaum, S. and Rosenblum, D. S. explore how uncertainty affects software testing [8]. Uncertainty in self-adaptive systems is explored in [4], [10], [12], [33], [47], [48]. Performance and reliability analysis under uncertainty is explored in [6], [23], [32], [42]. Uncertainty has been well studied in the field of formal methods. Probabilistic symbol model checkers such as PRISM [24] and LTSA-PCA [36] can deal with aleatory uncertainty. Three-valued logic consisting of `True`, `False`, and `Undefined` can represent epistemic uncertainty as in VDM (Vienna Development Method) [18]. As an empirical study, Ebert, F. et al. identify the factors that confuse code reviewers and understand how confusion impacts the efficiency and effectiveness of code reviews [7]. They do not make a distinction between lack of knowledge, confusion, or uncertainty. On the other hand, we focus on uncertainty itself.

As overviewed above, there are no empirical studies that systematically analyze the tendency of uncertainty in real software development projects.

B. Lessons Learned from Our Study

From the answer to RQ1, we can observe that uncertain program behavior, uncertain valuable / value / name, uncertain program defects are crucial to developer. It is necessary to provide the analysis tools for investigating the cause of such uncertain behavior, unexpected value, and uncertain bug. From the answer to RQ2, we can observe that some committers might take an action for not resolving but escaping or ignoring uncertainty. The reason of these actions is considered that uncertain issues are difficult to be resolved comparing to ordinary issues. We expect that uncertainty-aware tools can relax this problem. From the answer to RQ3, we can observe that uncertainty commits appear shortly after uncertainty-causing commits. At the same time, there are cases in which the code is repeatedly modified many times. It is preferable to provide the tools for managing why/when/where uncertainty arises or is fixed to be certain. It is also important to provide the tools for supporting uncertainty-aware requirements management, uncertainty-aware refactoring, and uncertainty-aware testing / verification. A part of the state-of-the-art research provides the tools for uncertainty-aware modeling, verification, and testing. This direction is favorable for developers suffering uncertainty. As the next step, we should provide an integrated development environment (IDE) for supporting all phases of software development in terms of uncertainty. To deal with this problem, we are developing an IDE called *iArch-U* [5] for embracing uncertainty in software development.

VII. Conclusion

Embracing uncertainty in software development is one of the crucial research topics in software engineering. Garlan, D. discusses the future of software engineering from the viewpoint of uncertainty [21]. He claims that *software engineering is founded on a computational myth that no longer fully serves its purpose: that the computational environment is predictable and in principle fully specifiable, and that the systems that compute in those environments can in principle be engineered so that they are trouble-free*. He argues that we must embrace uncertainty within the engineering discipline of software engineering. Although uncertainty has attracted a growing interest among researchers, there have been no empirical studies until now. Our empirical study is the first attempt to explore uncertainty in actual software

[5]The iArch-U IDE is open source software and can be downloaded from http://posl.github.io/iArch/.

development. As clarified in this paper, uncertainty exists everywhere in a certain percentage and developers cannot ignore the existence of uncertainty.

ACKNOWLEDGMENTS

We thank Takuya Fukamachi, Guan Da Jiang and Hokuto Muraoka for their great contributions. They were students of Naoyasu Ubayashi. This work was supported by JSPS KAKENHI Grant Numbers JP26240007.

REFERENCES

[1] Autili, M., Cortellessa, V., Ruscio, D. D., Inverardi, P., Pelliccione, P., and Tivoli, M., Integration Architecture Synthesis for Taming Uncertainty in the Digital Space, In *Proceedings of the 17th Monterey Conference on Large-Scale Complex IT Systems: Development, Operation and Management*, pp.118-131, 2012.

[2] Bornholt, J., Mytkowicz, T., and McKinley, K. S., Uncertain¡T¿: A First-Order Type for Uncertain Data, In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2014)* pp.5166 2014.

[3] Brown, C. E., Coefficient of Variation, *Applied Multivariate Statistics in Geohydrology and Related Sciences*, pp.155-157, 1998.

[4] Cheng, S. -W. and Garlan, D., Handling Uncertainty in Autonomic Systems, In *Proceedings of the International Workshop on Living with Uncertainties (IWLU 2007)*, 2007.

[5] Devanbu, P., Zimmermann, T., and Bird, C., Belief & Evidence in Empirical Software Engineering, In *Proceedings of the 38th International Conference on Software Engineering (ICSE 2016)*, pp.108-119, 2016.

[6] Devaraj, A., Mishra, K., and Trivedi, K. S., Uncertainty Propagation in Analytic Availability Models, In *Proceedings of the Symposium on Reliable Distributed Systems (SRDS 2010)*, pp.121-130, 2010.

[7] Ebert, F., Castor, F., Novielli, N., and Serebrenik, A., Confusion Detection in Code Reviews, In *Proceedings of 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME 2017)*, pp.549-553, 2017.

[8] Elbaum, S. and Rosenblum, D. S., Known Unknowns: Testing in the Presence of Uncertainty, In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*, pp.833-836, 2014.

[9] Eramo, R., Pierantonio, A., and Rosa, G., Uncertainty in Bidirectional Transformations, In *Proceedings of the 6th International Workshop on Modeling in Software Engineering (MiSE 2014)*, pp.37-42, 2014.

[10] Esfahani, N., Kouroshfar, E., and Malek, S., Taming Uncertainty in Self-Adaptive Software, In *Proceedings of the 8th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2011)*, pp.234-244, 2011.

[11] Esfahani, N., Razavi, K., and Malek, S., Dealing with Uncertainty in Early Software Architecture, In *Proceedings of the 20th International Symposium on the Foundations of Software Engineering (FSE 2012)*, pp.21:1-21:4, 2012.

[12] Esfahani, N. and Malek, S., Uncertainty in Self-Adaptive Software Systems, *Software Engineering for Self-Adaptive Systems II*, volume 7475 of LNCS, pp.214-238, Springer, 2013.

[13] Esfahani, N., Malek, S., and Razavi, K., Guidearch: Guiding the Exploration of Architectural Solution Space under Uncertainty, In *Proceedings of the 35th International Conference on Software Engineering (ICSE 2013)*, pp.43-52, 2013.

[14] Famelis, M., Salay, R., and Chechik, M., Partial Models: Towards Modeling and Reasoning with Uncertainty, In *Proceedings of the 34th International Conference on Software Engineering (ICSE 2012)*, pp.573-583, 2012.

[15] Famelis, M., Salay, R., Sandro, A. D., and Chechik, M., Transformation of Models Containing Uncertainty, In *Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013)*, pp.673-689, 2013.

[16] Famelis, M, Ben-David, N., Sandro, A. D., Salay, R., and Chechik, M., MU-MMINT: an IDE for Model Uncertainty, In *Proceedings of the 37th International Conference on Software Engineering (ICSE 2015)*, Demonstrations Track, pp.697-700, 2015.

[17] Famelis, M., Rubin, J., Czarnecki, K., Salay, R., and Chechik, M., Software Product Lines with Design Choices: Reasoning about Variability and Design Uncertainty, In *Proceedings of the 20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2017)* pp.93-100, 2017.

[18] Fitzgerald, J. and Larsen, G. P., *Modeling Systems, Practical Tools and Techniques in Software Development*, Cambridge University Press, 1998.

[19] Fukamachi, T., Ubayashi, N., Hosoai, S., and Kamei, Y., Conquering Uncertainty in Java Programming, In *Proceedings of the 37th International Conference on Software Engineering (ICSE 2015)*, Poster Track, pp.823-824, 2015.

[20] Fukamachi, T., Ubayashi, N., Hosoai, S., and Kamei, Y., Modularity for Uncertainty, In *Proceedings of the 7th International Workshop on Modelling in Software Engineering (MiSE 2015)*, pp.7-12, 2015.

[21] Garlan, D., Software Engineering in an Uncertain World, In *Proceedings of FSE/SDP Workshop on Future of Software Engineering Research (FoSER 2010)*, pp.125-128, 2010.

[22] Ghezzi, C., Sharifloo, A. M., Quantitative Verification of Non-functional Requirements with Uncertainty, *Dependable Computer Systems*, pp.47-62, 2011.

[23] Goseva-Popstojanova K. and Kamavaram S., Assessing Uncertainty in Reliability of Component-Based Software Systems, In *Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE 2003)*, pp.307-320, 2003.

[24] Hinton, A., Kwiatkowska, M., Norman, G., and Parker, D., PRISM: A Tool for Automatic Verification of Probabilistic Systems, In *Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2006)*, pp.441-444, 2006.

[25] Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., Germán, D. M., and Damian, D., The Promises and Perils of Mining GitHub, In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*, pp.92-101, 2014.

[26] Lago, P. and Vliet, H., Explicit Assumptions Enrich Architectural Models, In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, pp.206-214, 2005.

[27] Letier, E., Stefan, D., and Barr, E. T., Uncertainty, Risk, and Information Value in Software Requirements and Architecture, In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*, pp.883-894, 2014.

[28] Lindberg, C. A.(ed.): *Oxford American Writer's Thesaurus*, Oxford University Press, 2012.

[29] Llerena, Y. R. S., Dealing with Uncertainty in Verification of Nondeterministic Systems, In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*, pp.787-790, 2014.

[30] Marcus, M. P., Marcinkiewicz, M. A. and Santorini, B.: Building a large annotated corpus of English: The Penn Treebank, *Computational linguistics*, Vol.19, no.2, pp.313-330, 1993.

[31] Massey, A., Rutledge, R., Antón, A., and Swire, P., Identifying and Classifying Ambiguity for Regulatory Requirements, In *Proceedings of the 22nd International Requirements Engineering Conference (RE 2014)*, pp.83-92, 2014.

[32] Meedeniya, I., Moser, I., Aleti, A., and Grunske, L., Architecture-Based Reliability Evaluation under Uncertainty, In *Proceedings of the 7th International ACM Sigsoft Conference on the Quality of Software Architectures (QoSA 2011)*, pp.85-94, 2011.

[33] Perez-Palacin, D. and Mirandola, R., Uncertainties in the Modeling of Self-adaptive Systems: a Axonomy and an Example of Availability Evaluation, In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. pp.3-14, 2014.

[34] Ratnaparkhi, A.: A Maximum Entropy Model for Part-Of-Speech Tagging, In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 133-142, 1996.

[35] Raccoon and Dog, Unknownness, *ACM SIGSOFT Software Engineering Notes*, Volume 38 Issue 5, pp.8-17 , 2013.

[36] Rodrigues, P., Lupu, E., and Kramer, J., LTSA-PCA: Tool Support for Compositional Reliability Analysis, In *ICSE Companion 2014 Companion Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*, pp. 548-551, 2014.

[37] Rosenblum, D., Probability and Uncertainty in Software Engineering, Keynote Talk at the 2013 National Software Application Conference (NASAC 2013), http://www.slideshare.net/dsrosenblum/nasac-2013, 2013.

[38] Sommerville, I., Integrated Requirements Engineering: A Tutorial, *IEEE Software*, pages 16-23. IEEE, January/February, 2005.

[39] Salay, R., Chechik, M., Horkoff, J., and Sandro, A. D., Managing Requirements Uncertainty with Partial Models, *Requirements Engineering*, Volume 18, Issue 2, pp.107-128, 2013.

[40] Salay, R., Gorzny, J., and Chechik, M., Change Propagation Due to Uncertainty Change, In *Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering (FASE 2013)*, pp.21-36, 2013.

[41] Śliwerski, J., Zimmermann, T., and Zeller, A.: When Do Changes Induce Fixes?, *ACM Sigsoft Software Engineering Notes*, Vol. 30, No. 4, pp.15, 2005.

[42] Trubiani, C., Meedeniya, I., Cortellessa, V., Aleti, A., and Grunske, L., Model-Based Performance Analysis of Software Architectures under Uncertainty, In *Proceedings of the 9th International ACM Sigsoft Conference on the Quality of Software Architectures (QoSA 2013)*, pp.69-78, 2013.

[43] Ubayashi, N., Muraoka, H., Muramoto, D., Kamei, Y., and Sato, R., Exploring Uncertainty in GitHub OSS Projects, 40th International Conference on Software Engineering (ICSE 2018), Poster, to appear.

[44] Uchitel, S. Kramer, J., and Magee, J., Modelling Undefined Behaviour in Scenario Synthesis, In *Proceedings of the 2nd International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools at ICSE03*, 2003.

[45] Walker, W. E., Harremoës, P., Romans, J., van der Sluus, J. P., van Asselt, M. B. A., Janssen, P., and Krayer von Krauss, M. P., Defining Uncertainty. A Conceptual Basis for Uncertainty Management in Model-based Decision Support, Integrated Assessment, 4(1):5-17, 2003.

[46] Watanabe, K., Ubayashi, N., Fukamachi, T., Nakamura, S., Muraoka, H., and Kamei, Y.: iArch-U: Interface-Centric Integrated Uncertainty-aware Development Environment, In *Proceedings of the 9th International Workshop on Modelling in Software Engineering (MiSE 2017) (Workshop at ICSE 2017)*, pp.40-46 (2017).

[47] Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. C., and Bruel, J. -M., Relax: A Language to Address Uncertainty in Self-Adaptive Systems Requirement, *Requirements Engineering*, 15(2), pp.177-196, 2010.

[48] Yang, W., Xu, C., Liu, Y., Cao, C., Ma, X., and Lu, J., Verifying Self-Adaptive Applications Suffering Uncertainty, In *Proceedings of the 29th International Conference on Automated Software Engineering (ASE 2014)*, pp.199-210, 2014.

[49] Ziv, H., Richardson, D. J., and Klösch, R., The Uncertainty Principle in Software Engineering, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.8700, 1996.