Git-based Integrated Uncertainty Manager

1st Naoyasu Ubayashi  
Kyushu University  
Fukuoka, Japan  
ubayashi@acm.org

2nd Takuya Watanabe  
Edirium K.K.  
Tokyo, Japan  
sodium@edirium.co.jp

3rd Yasutaka Kamei  
Kyushu University  
Fukuoka, Japan  
{kamei, sato}@ait.kyushu-u.ac.jp

Abstract—Nowadays, many software systems are required to be updated and delivered in a short period of time. It is important for developers to make software embrace uncertainty, because user requirements or design decisions are not always completely determined. This paper introduces iArch-U, an Eclipse-based uncertainty-aware software development tool chain, for developers to properly describe, trace, and manage uncertainty crosscutting over UML modeling, Java programming, and testing phases. Integrating with Git, iArch-U can manage why/when/where uncertain concerns arise or are fixed to be certain in a project. In this tool demonstration, we show the world of uncertainty-aware software development using iArch-U. Our tool is open source software released from http://posl.github.io/iArch/.

Index Terms—uncertainty, management, trace, Git, IDE

I. INTRODUCTION

Uncertainty is a crucial problem in modern software development processes [4]. Software developers often face uncertainty in requirements elicitation, design decision, and selection of implementation alternatives. For example, they have to deal with the following situations: Case 1) uncertain which algorithm should be adopted to realize performance requirements; and Case 2) uncertain whether or not a code snippet is finally used because of changeable stakeholder requirements. These concerns arise not only in a programming phase but also in all software development phases. Nowadays, uncertainty appears repeatedly in daily software development activities, because many software systems have to be updated and delivered in a short period of time.

This paper introduces iArch-U, an uncertainty-aware software development tool chain, for software developers to not only describe uncertain concerns but also manage why/when/where they arise or are fixed to be certain. Our tool chain is provided as an Eclipse-based IDE (Integrated Development Environment).

There are two types of uncertainty: Known Unknowns and Unknown Unknowns. 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. 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.

The iArch-U IDE supports Known Unknowns based on the variability modeling consisting of alternative and optional. The former can describe uncertainty such as Case 1 and the latter can handle Case 2. Our approach is inspired from software product lines and has the expressive power equal to popular uncertainty representation method based on partial model [1]. Integrating iArch-U with the Git version control system, software developers can easily trace when and where uncertain concerns appear.

This paper is structured as follows. The technical background is briefly provided in Section II. The tool features of iArch-U are shown in Section III. We present a case study demonstrating the usefulness of iArch-U in Section IV. Concluding remarks are provided in Section V.

II. UNCERTAINTY DESCRIPTION AND MANAGEMENT

Uncertainty management in iArch-U is based on Archface-U [3], an architectural interface for uncertainty. This interface plays an important role in managing and tracing uncertainty crosscutting over UML modeling, Java programming, and testing phases. By imposing the same interface on UML models and Java programs, uncertainty in a program can be syntactically traced back to its origin, e.g. uncertainty appearing in a UML model.

Archface-U represents uncertainty in terms of architectural views consisting of both structural and behavioral aspects. The former can specify whether or not a method is needed (e.g. Case 2 in Section I) and the latter can specify alternative method behaviors (e.g. Case 1). Using Archface-U, we can add or delete uncertainty modularly as a constraint to a UML model or a program. This module mechanism is called Modularity for Uncertainty [3]. We regard Archface-U as a software module specifying uncertainty. This module is a management unit for uncertainty. By tracing the change history of Archface-U modules, we can systematically manage uncertainty in software development.

A. Archface-U: Interface for Managing Uncertainty

We explain the language features of Archface-U in detail. Archface-U, which supports component-and-connector architecture, consists of two kinds of interface: component and connector. The former declares a structural aspect, i.e. a class structure. The latter declares a behavioral aspects, i.e. coordination among components. Here, we use a simple example to explain the syntax of Archface-U as shown in Fig. 1. This example is a printer-scanner system, a well-known parallel system that falls into a deadlock [5]. Two processes
There are two aspects of Archface-U: constraints on UML models or Java programs containing uncertainty; and documentation for recording uncertainty. In the former aspect, localization is effective in tracing uncertainty crosscutting over even if uncertainty exists in models or programs. This precise localization is effective in tracing uncertainty crosscutting over models and programs. The constraint aspect of Archface-U is supported by the main iArch-U toolset shown in Section III-A. On the other hand, in the latter aspect, Archface-U is regarded as a document for recording uncertainty. We can trace when uncertainty appears or disappears by checking the modification history of Archface-U descriptions. The documentation aspect of Archface-U is supported by the Git-based uncertainty explorer shown in Section III-B.

### B. Bilateral Character of Constraints and Documents

In this section, we show the overview of iArch-U. We focus on uncertainty management integrating with Git.

---

**Fig. 1. Archface-U Description (Printer-Scanner System)**

| List 1 | 01: interface component cPrinter {  | 16: interface connector cSystem {  |
| | 02: public void put();  | 17: cCopyMachine F, cCopyMachine Q,  |
| | 03: public void print();  | 18: cPrinter printer, cScanner scanner (  |
| | 04:  }  | 19: GET = ({printer.get -> scanner.get,  |
| | 05:  }  | 20: PUT = (printer.put -> scanner.put);  |
| | 06: interface component cScanner {  | 21: COPY = (scanner.utility -> printer.print);  |
| | 07: interface component cScanner {  | 22: COPY = (scanner.scan -> printer.print);  |
| | 08: public void get();  | 23: }  |
| | 09: public void put();  | 24: P.copy = (GET -> COPY -> PUT -> P.copy);  |
| | 10: public void scan();  | 25: Q.copy = (GET -> COPY -> PUT -> Q.copy);  |
| | 11: interface connector cSystem {  | 26: }  |
| | 12: extends cSystem (  | 27: }  |
| | 13: interface component cCopyMachine Q,  | 28: extends cSystem (  |
| | 14: public void copy();  | 15: GET = {{printer.get -> scanner.get,  |
| | 15:  }  | 16: scanner.get -> printer.get));  |
| | 16: interface connector cSystem {  | 17: COPY = {{scanner.utility -> scanner.scan ->  |
| | 17: extends cSystem (  | 18: optional implementation  |
| | 18: cPrinter printer, cScanner scanner (  | 19: )  |
| | 19: GET = {{printer.get -> scanner.get,  | 20: }  |
| | 20: scanner.get -> printer.get});  | 21: PUT = {{printer.put -> scanner.put);  |
| | 21: COPY = {{scanner.utility -> scanner.scan ->  | 22: COPY = {{scanner.scan -> printer.print);  |
| | 22: COPY = (scanner.scan -> printer.print);  | 23: }  |
| | 23: }  | 24: }  |

---

**Fig. 2. iArch-U IDE**

**Fig. 3. Traceability among Archface-U, Model, and Code**

Archface-U is regarded as a syntactical language construct that is the target of compilation and verification. We can not only localize uncertain model/code region syntactically but also check whether an important property is guaranteed even if uncertainty exists in models or programs. This precise localization is effective in tracing uncertainty crosscutting over models and programs. The constraint aspect of Archface-U is supported by the main iArch-U toolset shown in Section III-A. On the other hand, in the latter aspect, Archface-U is regarded as a document for recording uncertainty. We can trace when uncertainty appears or disappears by checking the modification history of Archface-U descriptions. The documentation aspect of Archface-U is supported by the Git-based uncertainty explorer shown in Section III-B.
A. Overview of iArch-U

Fig. 2 shows the snapshot of iArch-U IDE consisting of UML-based model editor, Java program editor, uncertainty-aware compiler / model checker, and unit test support.

Model and Program Editor: Using our model editor, a developer can specify uncertainty in class diagrams and sequence diagrams. We slightly extend UML diagrams to represent alternative and optional. Archface-U can be automatically generated from these diagrams to reduce the cost of defining Archface-U manually. In iArch-U, Java programming based on Archface-U is supported by our editor.

Uncertainty-aware Compiler / Model Checker: The behavioral correctness of models and programs is guaranteed modularly using our compiler integrating type checker and model checker. The type checker based on the refinement calculus focusing on simulation checks the conformance between Archface-U and its model/code. The traceability between models and programs can be also guaranteed via Archface-U type checking as illustrated in Fig. 3. That is, uncertainty can be systematically managed crosscutting over design and coding phases. The model checker verifies the behavioral properties such as a deadlock by only using the information described in Archface-U. Integrating type checker and model checker, we can verify behavioral properties at the model/code level uncertainty because of a simulation relation not only between a model and its Archface-U but also between a program and its Archface-U. When both type checking, for a model and for code, are passed, they are consistent and traceable.

Unit Test Support: Integrating with JUnit and AspectJ, we can execute runtime unit testing when we want to check properties that cannot be checked statically. Test case drivers are automatically generated as aspects from Archface-U. By weaving an aspect, we can test each alternative.

B. Git-based Uncertainty Explorer

Using Git, we can manage why/when/where uncertain concerns arise or are fixed to be certain in design/coding/testing phases as shown in Fig. 4 and Table I. Archface-U is not merely an interface but a document for describing uncertain concerns. In iArch-U IDE, a branch is automatically created and a message input dialogue is shown for a committer to write the reason why an uncertain concern is defined in Archface-U. When the uncertain concern is fixed to be certain and the uncertain Archface-U is merged to an original Archface-U, the branch is automatically merged to the master.

When some uncertain concerns are resolved during the design or coding process, the trace of this uncertainty only remains in the commit history. It is troublesome to check the historical changes of uncertainty from raw Java code commits. The iArch-U IDE provides a feature to mine the change history of Archface-U descriptions from a Git repository and to display the summary of uncertainty changes as shown in Fig. 5. Basic information for each uncertainty, date of the latest uncertainty related commit and how the uncertainty changed its state, is available.

C. How to Manage Uncertainty

By introducing the simple tool automation shown above, we can enjoy the valuable properties below. Each number corresponds to the number in Table I.

1) The reason for appearing uncertainty can be obtained from a commit comment written by a committer who modifies Archface-U descriptions.
2) A developer can easily know the time when an uncertain concern arises and is fixed to be certain by looking at the timestamp of committing Archface-U descriptions.
3) The location of uncertain code snippets is traced by looking at Archface-U descriptions.
4) The person who finds uncertainty is obtainable by checking the committer name.
5) We can trace which code region is optional or alternative only by looking at Archface-U descriptions.
Whether or not undo memory should be included here is debatable.

Type checking plays also an important role in the uncertainty management. An *Archface-U* description is not an informal document but a formal specification which can be verified whether the description does not contain inconsistency and the code conforms to the *Archface-U*. We can trace the uncertain code region correctly, because the region is described in *Archface-U*. If there is not a type checker, the location of the code region is not guaranteed. Uncertainty management based on type checking approach is not only simple but also effective in tracing the life-cycle of uncertainty. If uncertainty is not described in an interface such as *Archface-U*, it might be difficult to automate the management of uncertainty.

D. Implementation

The *iArch-U* IDE is implemented as an Eclipse plug-in. *Archface-U* is defined as a DSL (Domain-Specific Language) by using Xtend. Type checker is implemented by using APIs for analyzing AST (Abstract Syntax Tree) provided in JDT (Java Development Tools). Model editor is implemented by using EMF (Eclipse Modeling Framework) and Graphiti (Graphical Tooling Infrastructure). LTSA (LTS Analyzer) is used as the uncertainty-aware model checking engine, because *Archface-U* is based on FSP supported by LTSA.

IV. Case Study

We show the effectiveness of *iArch-U* using the GIMP (GNU Image Manipulation Program) project as a case study.

A. Uncertainty in GIMP Project

Table II picks up impressive commit messages in the GIMP project. The phenomena suffering from uncertainty is not uncommon, because many developers have participated in the GIMP project relatively recently in the history of its development (No.1). Some bugs are related to their execution context (No.2). It is not easy for a current developer to decide whether a method or a code snippet written by old developers are really needed (No.3). Fig. 6 shows the detailed comment of No.1 and the difference between the modified code and the original. In this case, the developer removed three functions although it is not certain whether they are really unnecessary. Maybe this removal was correct decision, because the issues related to this commit did not arise.

This case study implies that many developers tend to have a difficulty for tracing when and why an uncertain concern arises and is resolved. The *iArch-U* IDE is helpful for developers to manage these uncertainties.

B. Applicability of Our Approach

The uncertain concerns shown in Table II can be relaxed by using optional. That is, we can move unknown, unclear or unsure methods to optional methods. For example, the unknown functions in Fig. 6 can be described as follows.

```java
[gimp_transform_tool_recalc();]
[gimp_transform_tool_prepare();]
[gimp_transform_tool_bounds();]
```

If an old developer involved in the GIMP project specified uncertainty as above, the committer of Fig. 6 did not have to worry about the unknown reasons and could trace when and why this uncertainty occurred. Moreover, the committer could verify whether a program behaved correctly even if these three functions were removed.

V. Conclusions

The *iArch-U* IDE is open source software and everyone can experience the world of uncertainty-aware software development. The manuals, tutorial documents, and promotion videos can be downloaded from http://posl.github.io/iArch/. In this demonstration, we not only illustrate the design concept of *iArch-U* but also show how to manage uncertainty in design, coding, and testing phases by using a practical example.

Acknowledgments: We thank Takuya Fukamachi, Shunya Nakamura, and Keisuke Watanabe for their great contributions. They were students of our research group. This work was supported by JSPS KAKENHI Grant Numbers JP26240007 and 18H04097.

REFERENCES