Git 連携による不確かさマネジメントシステム

深町 拓也[†1,a)]　鵜林 尚靖[†1,b)]　細合 晋太郎[†1,c)]　亀井 靖高[†1,d)]

概要：ソフトウェア開発において，曖昧なユーザの要求や仕様がよく分かっていない API 等の「不確かさ」は避けることが出来ない問題である．我々はこのような不確かさを明示的に記述できるインターフェース機構 *Archface-U* を提案してきた．また，我々はこのインターフェース機構と Git を用いて不確かさをマネジメントする方法も提供してきた．本論文では，このマネジメント機構を用いて不確かさの記録と追跡を効率的に行うことができるツールの実装概要を示す．

1. はじめに

ソフトウェア工学において，不確かさを包容したソフトウェア開発は重要な研究課題の 1 つである [21,25,26,29,36]．Garlan は，不確かさという観点から将来のソフトウェア工学について論じた [15]．彼は，「コンピュータ環境が予測可能であり原則として完全に明記でき，また，そういった環境下で動くシステムは障害が無いように設計されている，という神話に基いてソフトウェア工学の研究は行われている」と主張しており，ソフトウェア工学の領域に不確かさを包容させなければならないと述べている．

ソフトウェア開発における不確かさには 3 つのタイプ（*Known Knowns*，*Known Unknowns*，*Unknown Unknowns*）がある [5]．*Known Knowns* タイプは，不確かさが存在しない開発であり，多くの伝統的なソフトウェア工学の研究において研究が行われている．*Known Unknowns* タイプは，ソフトウェア開発プロセス中に不確かな問題が存在する開発である．しかし，*Known Unknowns* タイプでは，この不確かな問題は開発者や消費者といったステークホルダー間で認識，共有されている．例えば，複数のどれが選ばれるかわからない要求がこのタイプの不確かさに当たる．一方で，*Unknown Unknowns* タイプは，何が不確かであるかがわからないケースを指す．つまり，このタイプを解決することは難しく，どのような問題が発生するかが予測不可能である．

しかし，*Knowns Unknowns* タイプの不確かさであっても，従来，開発者はこれをリスク管理の一つとして扱い，

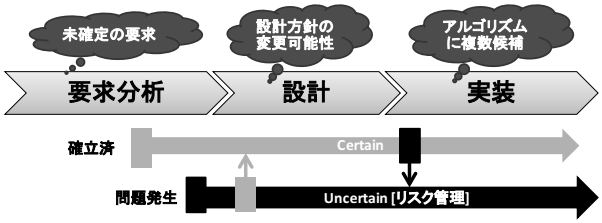

図 1: ソフトウェア開発における不確かさとその従来の対処

リスク管理表や，仕様書などにメモとして記載するなどの対処をしてきた (図 1)．

しかし，このような対処による記述は自然言語に依存するため，記述している不確かさが実装やテストのソースコードや設計モデルのどの箇所に存在しているのかが分からなくなることが多い．そこで，我々は従来より，不確かさを適切に記述し，不確かさをコード上に抱えていても，実装を進めることができるインターフェース機構として *Archface-U* を提案してきた [37]．

また，我々はこのインターフェースを用いて不確かさをマネジメントする方法を提案してきた [38]．本論文では，このマネジメントをサポートするツールの概要とその実装方法を具体的に示すことによって，不確かさが従来のリスク管理に比べてより適切に扱えることを明らかにする．具体的には，不確かさが発生する状況，それに対応するためのシナリオを例示し，*Archface-U* とバージョン管理システムである Git [16] を用いて不確かさが適切にマネジメントできることを示す．また，*Archface-U* の開発環境である *iArch-U* の機能を用いてこのマネジメントをどのようにサポートするのかを具体的に示す．

本論文において，2 章では，本研究で扱う不確かさを定義するとともに，従来の不確かさのマネジメントの問題点を提示し，本研究におけるアプローチを述べる．3 章では，

†1 現在，九州大学
Presently with Kyushu University
a) fukamachi@posl.ait.kyushu-u.ac.jp
b) ubayashi@ait.kyushu-u.ac.jp
c) hosoai@qito.kyushu-u.ac.jp
d) kamei@ait.kyushu-u.ac.jp

表 1: Perez-Palacin らによる不確かさの分類

観点	性質	性質の説明
場所	コンテキスト	環境に関する不確かさ
	モデル構造	モデル自体の構造に現れる不確かさ
	入力パラメータ	モデルへの入力パラメータに関する不確かさ
レベル	レベル 0	確定している知識
	レベル 1	知識の不足を認知している状態．既知の不確かさ．
	レベル 2	知識の不足を認知できていない状態．未知の不確かさ．
	レベル 3	不確かさを認知するプロセス自体が不足している状態．
	レベル 4	メタな不確かさ．不確かさのレベル自体が不確か．
性質	認識的	十分なデータや知識が無いために発生する不確かさ
	偶発的	物理現象等の確率的不確かさ

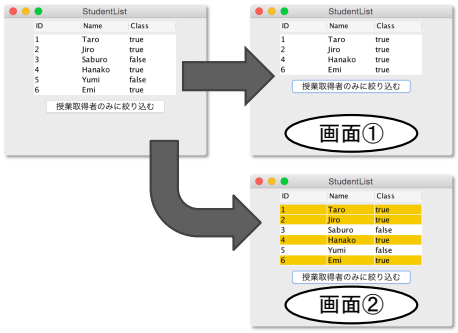

図 2: GUI アプリケーションにおける不確かさの例

本開発プロセスで用いるインターフェース機構 *Archface-U* について説明する．4 章では，*Archface-U* と Git を用いた不確かさのマネジメントの方法をツールの利用法とともに示す．5 章では，*Archface-U* の開発環境である *iArch-U* によって，4 章で示したマネジメントをどのようにサポートするかを示す．6 章では，不確かさに関する関連研究と本研究との比較を行う．最後に，7 章で本研究のまとめを述べる．

2. 不確かさのマネジメントにおける課題

本章では，関連研究をもとに本研究における不確かさがどのようなものかを具体例とともに定義する．その後に，従来の不確かさのマネジメントにおける課題を述べ，その課題を解決するために行う本研究のアプローチを示す．

2.1 本研究における不確かさ

Perez-Palacin らは，不確かさについて，「場所 (*Location*)」，「レベル (*Level*)」，「性質 (*Nature*)」の 3 つの観点から分類をすることができるとしている [23](表 1)．本研究においては，表 1 における不確かさの分類のうち，以下の不確かさについて扱う．

- 場所による分類 − コンテキスト，モデル構造，入力パラメータ
- レベルによる分類 − レベル 1，レベル 0
- 性質による分類 − 認識的

上記のような不確かさを扱う理由としては，本マネジメントは，従来，リスク管理表等で管理されていた「現在分かっている不確かさ」をある程度自動化して扱うことを目的としているためである．

2.2 不確かさの具体例

本節では，2.1 節で定義した不確かさの具体例を上げ，我々が扱う不確かさについて明確に説明する．

この具体例では，ある教師が担当しているクラスの生徒の内，教師が担当している教科を受講している生徒を分かるようにする GUI アプリケーションを考える．この GUI アプリケーションは開発当初はボタンをクリックすると，受講者のみを表示するように生徒のリストが切り替わる (画面①) 仕様であった．ここで，依頼主である教師が受講をしていない生徒をリスト上から消さずに，受講をしている生徒に色をつけて表示する (画面②) とどのようになるかを見たいと希望したと仮定する．この時，最終的にこのアプリケーションの GUI が画面①，②のいずれになるかは実装後に教師に画面を表示するまで分からないという不確かさが発生する (図 2)．これは教師の要求により，現在開発者が把握ができている「不確かさ」であるといえる．

2.3 従来の不確かさのマネジメントにおける課題

本節では，2.1 節のような不確かさをマネジメントする上で抱えている問題点をあげる．

2.3.1 不確かさの形式的な記述

2.1 節のような不確かさは実装上の方針がぶれたり，設計の見直しが必要となるようなものであるが，ソースコード上や UML モデル上等にこのような不確かさを適切に記述するのが難しい．このような非形式的な不確かさの記述は，プロジェクトを Git で管理していたとしても，その不確かさがソースコードのどの部分にあたるのかをログを見ながら確認する必要があり，確実性と効率性に欠ける．

2.2 節における例であれば，GUI を画面①か画面②になるかが不明という不確かさが形式的に記述ができず，リスク管理表等の自然言語で記述するほかないといった問題がある．

2.3.2 現在あるいは過去の不確かさの状況の把握

現在あるいは過去において，不確かさがどのように設計や実装がされているかが分かりにくいという問題がある．Github の Issues 等の課題管理システムは，Git のコミットと自然言語で記述された課題を結びつけることが可能であり，コードの差分によって課題の解消の遷移や課題が起こった原因となるコミットを特定することができる．ただし，実際にプログラマはどのメソッドについて問題が起きているかに興味があり，これもやはり Git のログを辿り，ソースコードの差分を確認する必要がある．

例えば，2.2 節における例であれば，画面①の実装を行っているメソッドはどれで，そのメソッドをどのメソッドと

置き換えて画面②に変化させようとしているのか，といったことを端的に把握できない，といった問題がある．

2.4 本研究のアプローチ

2.3 節で述べた課題を解決するために，我々は 2.1 節の不確かさを記述することができる *Archface-U* というインターフェースを用いることを提案する．本マネジメントでは，ソフトウェア開発において，不確かさを発見した際，このインターフェースに記述をし，不確かさが解決したらその記述を削除することにより不確かさを表現し，そのログをバージョン管理システムである Git に残すことによって後から不確かさの遷移を追跡できるようにする．

3. 不確かさを包容するインターフェース *Archface-U*

本章では，*Archface-U* を紹介する．このインターフェースは過去に提案済み [37] であるため，本章では今回紹介するマネジメントに必要な説明を簡単に行う．

3.1 概要

Archface-U は設計と Java による実装の間のギャップを埋めるためのインターフェース機構である *Archface* [31] を不確かさが表現できるように拡張したものである．

Archface-U は Component-and-Connector アーキテクチャ [1] に基づいたインターフェース機構である．*Archface-U* において Component は，クラスおよびメソッドを宣言し，Connector は，メソッドの呼び出し関係を記述する．開発者はプログラムがインターフェースである *Archface-U* に従うように実装することにより，設計と実装をつなぐことができる．プログラムが *Archface-U* に従っているか否かは *Archface-U* コンパイラの型検査によって知ることができる．

型検査とは，*Archface-U* の Component や Connector のインターフェース記述を，Java コードのメソッドの定義や呼び出しが実装しているかを *Archface-U* コンパイラによって検査することである．インターフェース記述に従わない実装を行っている場合，型検査違反として *Archface-U* コンパイラはコンパイルエラーを返す．

3.2 不確かさを含んだインターフェース記述

Archface-U は確立した規約を記述する *Certain Archface* と，不確かな規約を記述する *Uncertain Archface* の 2 つのインターフェース記述によって構成される．

Uncertain Archface 内には以下 2 種類の既知の不確かさを記述することができる．

(1) いくつか Component の候補があるが，その中でどれを実際にシステムに組み込むかがわからないという不確かさ

(2) ある Component について，実際にシステムに組み込むかがわからないという不確かさ

この 2 種類の不確かさのうち，(1) を *Alternative*，(2) を *Optional* と以下呼称する．*Archface-U* 内では *Alternative* を { }，*Optional* を [ ] でメソッドを囲うことによって表現できるようにしている．リスト 1 に，2.2 節の例における *Archface-U* のインターフェース記述を示す．

```
1  interface component Main{
2    void actionPerformed(ActionEvent e);
3    void log(Student s);
4  }
5
6  interface component StudentController{
7    void filterStudent(JTable table);
8  }
9  uncertain component uStudentController
10 extends StudentController{
11   [void colorStudent(JTable table);]
12 }
13
14 interface connector cStudent{
15   Main = (Main.actionPerformed
16   -> Main.log -> Main);
17 }
18 uncertain connector ucStudent
19 extends cStudent{
20   Listener = (Main.actionPerformed ->
21   {uStudentController.colorStudent,
22   StudentController.filterStudent} -> Main);
23 }
```

リスト 1: GUI アプリケーションにおける *Archface-U* によるインターフェース記述の例

リスト 1 の Component インターフェースでは，実装上の `Main` クラスや `StudentController` クラスに対して各メソッドの宣言について記述している．例えば，`StudentController` クラスにおいて，`filterStudent` メソッドはコード上で宣言される必要があるが，`colorStudent` メソッドは *Optional* であるため宣言しても，しなくてもよい．

また，Connector インターフェースは，FSP(Finite State Process) [20] をベースとした記法によりメソッドの呼び出し関係を記述している．例えば，`actionPerformed -> log` という表記は `actionPerformed` メソッドにおいて `log` メソッドが呼び出されていることを表す．ここで，`uncertain connector` である `ucStudent` に着目する．`colorStudent`，`filterStudent` が *Alternative* として定義されているため，実質 `actionPerformed -> colorStudent` あるいは `actionPerformed -> filterStudent` のいずれかのメソッド呼び出し関係が成り立てばよい．なお，*Archface-U* は Famelis らの既存研究である，不確かな設計モデル Partial Model [12] とほぼ同等の内容を記述することが可能である [14]．

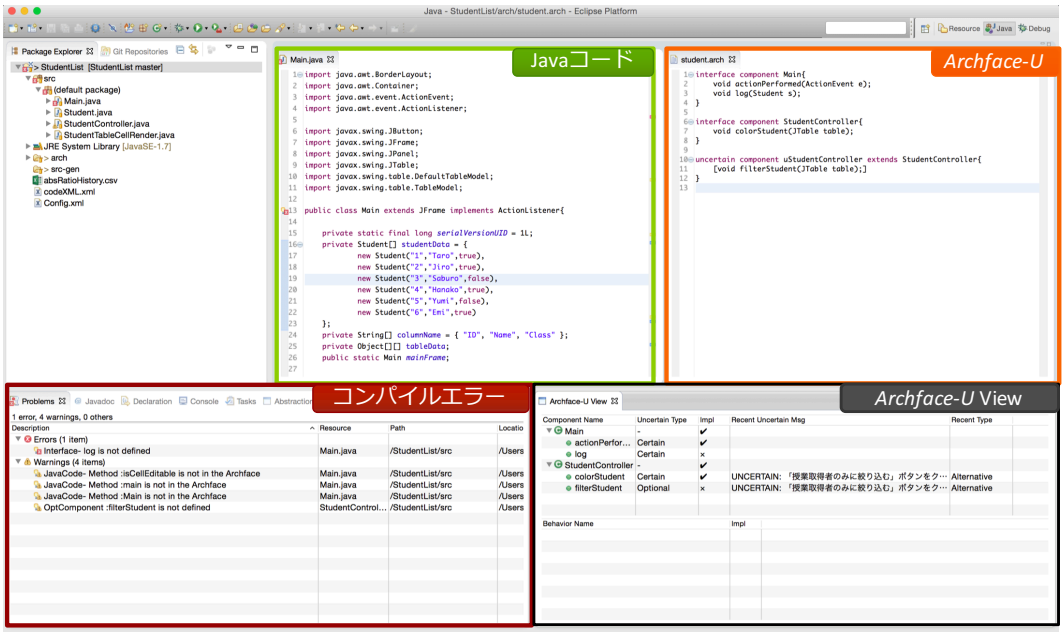

図 3: *iArch-U* のスナップショット

3.3 開発支援環境 *iArch-U*

不確かさを包容した開発を支援するために，我々は，*Archface* の開発環境である *iArch* [39] を拡張し，*iArch-U* という開発支援ツールを考案した．開発ツール *iArch-U* は *Archface-U* のコンパイラとそのエディタで構成されている．図 3 は *iArch-U* のスナップショットである．

Archface-U エディタは，統合開発環境 Eclipse 上で，*Archface-U* の記述支援を行う．*Archface-U* コンパイラは Java コードのコンパイルが行われると同時に *Archface-U* をコンパイルし，型検査を Java コードに対して行う．もし，型検査違反がある場合はそれをコンパイルエラーとして返す．さらに，*Alternative* や *Optional* といった不確かな制約において，現在の実装がどのような状態になっているかの情報を示す．具体的には，リスト 1 の例において `colorStudent` は定義してもしなくても良い *Optional* なメソッドであるが，これが現在定義されているかどうかを示すことができる．

4. 不確かさのマネジメント

本章では，3 章で述べた *Archface-U* と，バージョン管理システムである Git，及び *Archface-U* 用の統合開発環境である *iArch-U* を用いて不確かさをどのようにマネジメントするかを具体例を用いつつ述べる．

4.1 マネジメントの概要

今回提案する不確かさのマネジメント法では，*Archface-U* と Git を用い，不確かさを管理する．この方法では，不確かさを記録，追跡することによって不確かさをマネジメントする．具体的には，不確かさをマネジメントしたいプロジェクトを Git で管理し，不確かさが発見されたり，解消されたりした場合にその旨を *Archface-U* に記述しコミットする (図 4)．

4.2 ツールサポート

Archface-U の統合開発環境 *iArch-U* は 3.3 節における機能のほか，4.1 節に示した不確かさの記録や，不確かさの追跡を補助する機能を実装した．

なお，*iArch-U* は，Eclipse [4] のプラグインとして実装されている．アーキテクチャとしては，*Archface-U* を XText [33] を用いた DSL(Domain Specific Languages) として定義している．また，Java のソースコードを JDT(Java Development Tools) によって提供されている AST(Abstract Syntax Tree) 解析 API で解析することによって第 3.3 節にて述べた型検査を行う．型検査による違反は，コンパイル時に通常の Java ソースコードのコンパイルエラーとともに Eclipse 上に出力する．また，Git のログの解析に Git の Java ライブラリである JGit [17] を用い，Git のログをモデル化している．

4.3 不確かさの記録

本マネジメントにおいては，不確かさが要求，設計モデル，ソースコード等に見つかった場合，あるいはすでに把握していた不確かさが解消された場合その旨を *Archface-U* に記述することによって記録する．具体的には，不確かさの該当箇所である *Archface-U* の Component や Connector を *Optional* や *Alternative* に変更することによって表現する．以下では，2.2 節の例を用いて本マネジメント法を用いて不確かさを記録するシナリオを示す．

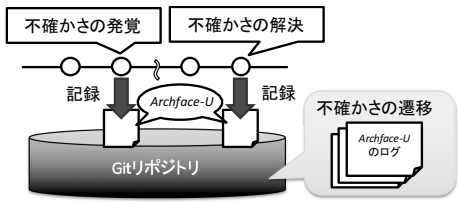

図 4: 本研究における不確かさのマネジメントの概要

この例において，不確かさは画面①と画面②のいずれにするかを決めることが出来ないというものである．この時，StudentController クラスにおいて，生徒のリストを講義の受講者のみにするためのメソッドを filterStudent，受講している生徒に色をつけるためのメソッドを colorStudent と定めたとする．この時，不確かさが発生する前の *Archface-U* はリスト 2 のように記述することができる (図 5 コミット c1)．

```
1  interface component Main{
2    void actionPerformed(ActionEvent e);
3    void log(Student s);
4  }
5
6  interface component StudentController{
7    void filterStudent(JTable table);
8  }
9
10 interface connector cStudent{
11   Main = (Main.actionPerformed
12   -> Main.log -> Main);
13   Listener = (Main.actionPerformed ->
14   StudentController.filterStudent -> Main);
15 }
```

リスト 2: 不確かさが発生する前の GUI アプリケーションにおける *Archface-U* によるインターフェース記述の例

ここで，教師の要求によって先述の GUI に関する不確かさが発生したとする．開発者は *Archface-U* に colorStudent によって機能を実装するかもしれないという不確かさを記述し，コミットを行う (図 5 コミット u1)．コミット u1 における *Archface-U* はリスト 1 のように記述できる．その後，colorStudent の実装を行いながら通常の Git のようにコミットしながら開発を進める．最終的に，プロトタイプが完成し，教師に確認してもらい，画面②の GUI に確定したとする．この時，開発者は画面②用に実装を行い，*Archface-U* の colorStudent を Certain にし，filterStudent の記述を *Archface-U* から削除し，コミットを行うことによって不確かさの解消を表現する (図 5 コミット c2)．

また，*iArch-U* は図 5 に示すような不確かさに関連するコミットをテンプレート化するために，「不確かさの概要」「不確かさの理由」「不確かさの影響範囲 (要求，設計，実装)」の記述を促す．これは，*Archface-U* に不確かさに関連する変更があった際にダイアログを出力することによって実現する．

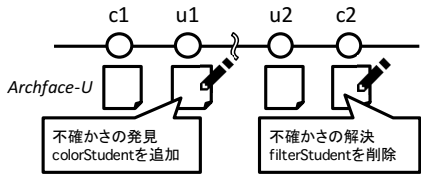

図 5: GUI アプリケーションにおけるマネジメントの例

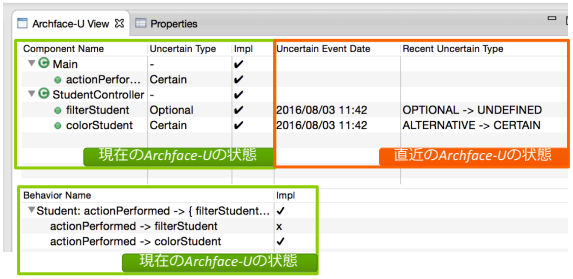

図 6: *Archface-U View* のスナップショット

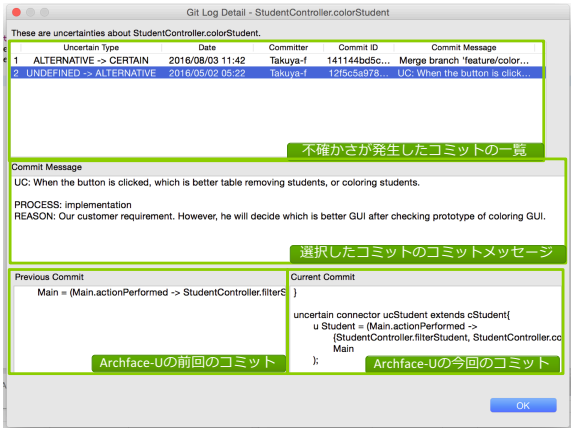

図 7: *Git Log Detail* ダイアログのスナップショット

4.4 不確かさの追跡

本サポート機構では，不確かさの追跡を Git のコミットログとそれに付随するコミットメッセージ，および *Archface-U* を観察することによって追跡できる．観察する各々において，以下の様な情報を取得できる．

- コミットログ： 不確かさの発生・解決日時，発見・解決者，不確かさが存在している・存在していた成果物
- コミットメッセージ： 不確かさの発生・解決理由，発生・解決した開発工程
- ***Archface-U***： 不確かさが存在するメソッドやクラス，不確かさの種別

なお，*Archface-U* の不確かさの種別とはその不確かさが Component か Connector のものか，また *Alternative* か *Optional* かを示す．

2.2 節における例において不確かさの追跡を行うことを考える．図 5 コミット u2 後，教師が再び GUI を画面①に戻してほしいという要求が発生したとする．もし，開発者が画面①を実装した開発者でない場合，Git のコミットメッセージや差分をマニュアルで追跡する必要がある．しかし，4.3 節のように不確かさについてのログを

Archface-U において残しておけば，ボタンのリスナである `actionPerformed` を含む Connector の *Archface-U* の差分のみを追跡すればよい．そうすることにより，画面①を実装していた `filterStudent` というメソッドを見つけることができる．*iArch-U* では，このような不確かさの追跡を手軽に行うために，*Archface-U View* と，*Git Log Detail* ダイアログを用いることができる．*Archface-U View* では直近の不確かさに関するコミットの日時とどのような不確かな状態からどのような不確かな状態にメソッドが遷移したか，を確認できる．より詳細な不確かさに関する情報は，*Archface-U View* の項目をダブルクリックすることによって表示される *Git Log Detail* ダイアログによって確認ができる (図 7)．ここでは，メソッドに関する全ての不確かさに関する変更が行われたコミットを表示し，それぞれのコミットメッセージやコミッタ，日時，不確かさの遷移を確認できる．

また，不確かさはコミット単位で記録されているため，該当コミットにおいてブランチを作成し，機能を変更したいブランチへマージを行うことによって `filterStudent` を適用することができる．

4.5 *Archface-U* の型検査による成果物の保証

4.3 節，4.4 節で記録，追跡されるコミットに付随する成果物は *Archface-U* に従うことが型検査によって保証される．これは Git 単体でコミットメッセージ等で不確かさを管理することとの差異である．

例えば，2.2 節における例の不確かさの追跡を考える．この時，図 5 コミット u2 においては，*Archface-U* はリスト 1 のように記述されるが，例えば，`Listener` の振る舞いの定義により，`actionPerformed` 内では `filterStudent` か `colorStudent` のいずれかが呼び出されていることがソースコード内でも保証される．

5. 適用事例

本章では，*Archface-U* を用いて，GIMP(GNU Image Manipulation Program) における不確かさをマネジメントすることによって，現実のソフトウェアプロジェクトにおいても *Archface-U* を用いたマネジメントが有効であることを示す．実際に本論文で示した手法を用いて不確かさのマネジメントを試みる．

ここでは，我々が以前行った GIMP 用いて行った実証分析 [34] をもとに事例を示す．この実証分析では，コミットメッセージから不確かさに関連する用語を抽出し，不確かなコミットを検出し，その結果に関して考察している．

5.1 GIMP における不確かさ

GIMP のような OSS の開発は，Github 等のホスティングサービスを用いることが多いが，このようなホスティングサービスでは Pull Request と呼ばれる仕組みを用いて，分散している開発者 (Contributors) のソースコードの変更をプロジェクトに取り込むかどうかを権限者 (Commiters) が承認して取り込む．しかし，この際，どの実装が不確かのまま承認されたか，あるいは非承認とされたか，といったことを簡潔に表すことは難しく，自然言語に頼る部分も多かった．

図 8: GIMP における不確かさを含むコミットとコミットメッセージ

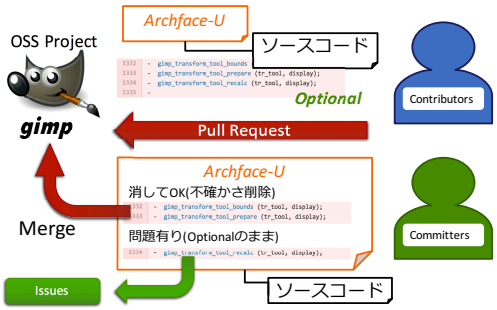

図 9: GIMP における図 8 の不確かさのマネジメント例

例えば，図 8 に示した事例 *1 では，一部の関数呼び出しが消去されると問題があるため削除ができず，Commiters がそれらの関数呼び出しを残してマージすることもありうる．このような場合，Pull Request によってマージされたコミットはいくつかの関数が削除されたことは差分によって知ることができるが，それ以外の消去されようとしていた関数については知ることが難しい．

5.2 GIMP の不確かさに対するマネジメント

我々が提案するマネジメントでは，5.1 節に示したような複雑な不確かさに関しても，以下の (1),(2),(3) の手順によって表現をし，記録することが可能である (図 9*2)．

(1) Contributors は削除したい関数呼び出しを *Optional* として表現し，ソースコード上では削除をして Pull Request を送信する．

(2) Commiters は削除して良い関数呼び出しを *Archface-U* から削除し，残さなければならない関数呼び出しを *Optional* にしたままにしてコミットをする．(このコミットを C_u とする．) また，*Optional* にした関数呼び出しはソースコード上では残す．

*1 `https://github.com/GNOME/gimp/commit/f078a7416c163e743bd19f6f5c0a250a08e8c4c8`

*2 説明の単純化のため図 9 及びリスト 3 については，`gimp_transform_tool_recalc` のみを対象としている．

（3） *Optional* にした関数は将来消される可能性があるため，プロジェクト全体に Issue として C_u に紐付けて残す．

以上の手順で不確かさの記録を行う場合，C_u にはリスト 3[*2] のような *Archface-U* が記述される．

```
1 uncertain connector uTransForm extends TransForm{
2   Recalc = (gimp_transform_tool_transform
3   (GimpTransformTool *tr_tool,
4   GimpDisplay *display) ->
5   [gimp_transform_tool_recalc
6   (GimpTransformTool *tr_tool,
7   GimpDisplay *display)] -> Recalc);
8 }
```

リスト 3: C_u における *Archface-U* 記述の一例

このような記録を行ったプロジェクトのソースコードを *iArch-U* を用いて開発を行えば，C_u において削除されようとしていた関数が *Archface-U View* によってすぐに確認をすることができる．もし，この関数の作者が消してはならないと判断すれば，*Archface-U* の *Optional* 記述を削除した Pull Request を送ることで不確かさを解消することも可能である．また，削除をされた関数についても，Pull Request のログから *Archface-U* のみに候補を絞り，検索をすることでどのような関数がどのコミットで削除をされたかということを *Archface-U* の変遷によって簡潔に追跡することができる．これらは，*Archface-U* というインターフェースを用いて不確かさをモジュール化して管理することができる，ということを示しており，極めて有用であると言える．

6. 関連研究

本節では，不確かさに関する研究動向について述べる．近年，不確かさは研究者に非常に注目されている [2,3,5–13,18,19,22–24,27,28,30,32,35]．

この中でも代表的な不確かさの研究として，Partial model を用いて *Known Unknowns* タイプの不確かさを表現する手法が Famelis らによって提案されている [11,12]．Partial model は，システムの全ての可能性のある設計を含む 1 つのモデルであり，そのモデルは命題論理へとエンコードされる．Partial model は，要求モデルにおける不確かさの特定，不確かさを減らすことによるモデルの改善，不確かさを含むモデル間の関係のトレーサビリティへの意味の付与，そして関係するモデル間での不確かさを減らす変更の波及などの分野で用いられている [27]．その他，不確かさに関する研究テーマは，ソフトウェアアーキテクチャ，モデル変換，テスト，検証，パフォーマンス工学，自己適応システムなど多岐にわたる (表 2)．

このように，最新の研究において不確かさの様々な側面を扱っているものの，不確かさのマネジメントに関する研究は無い．しかしながら，5 章において述べたように，GIMP のような実プロジェクトにおいて不確かさを起因とした問題が起きており，不確かさのマネジメントは必要であることが分かる．本研究はこの問題を解決するための一手法となると思われる．

表 2: 不確かさに関する研究

No.	研究トピック	リファレンス
1.	Modeling and Reasoning with Uncertainty	[12], [11]
2.	Uncertainty in Requirements	[18], [27]
3.	Uncertainty in Software Architecture	[10], [8], [18]
4.	Change Propagation Due to Uncertainty Change	[28]
5.	Model Transformation Containing Uncertainty	[6], [13]
6.	Testing or Verification in the Presence of Uncertainty	[5], [19]
7.	Self-Adaptive Applications Suffering Uncertainty	[2], [7], [9], [23], [32], [35]
8.	Performance and Reliability Analysis under Uncertainty	[3], [24], [22], [30]

7. おわりに

本論文では，*Archface-U* と Git を用いた不確かさのマネジメント法を提案した．また，そのマネジメントを開発環境 *iArch-U* によってどのようにサポートするかを示した．

今後の研究方針としては，現在リスク管理によく使われている課題管理システムと連携させることによって，ソフトウェア開発における課題を不確かさという単位でより扱いやすくすることができると考えられる．具体的には，*Archface-U* に不確かさが追加，削除された場合に課題を追加，解決するような操作や，課題管理システム上に *Archface-U* を表示してどのような不確かさが追加されたかを追跡しやすくする等の連携が考えられる．

謝辞

本研究は，文部科学省科学研究補助費基盤研究（A）（課題番号 26240007）による助成を受けた．

参考文献

[1] Allen, R. and Garlan, D.: Formalizing Architectural Connection, *Proceedings of the 16th International Conference on Software Engineering*, pp. 71–80 (1994).

[2] Cheng, S.-W. and Garlan, D.: Handling uncertainty in autonomic systems, *International Workshop on Living with Uncertainties (IWLU'07)*, p. 2007 (2007).

[3] Devaraj, A., Mishra, K. and Trivedi, K. S.: Uncertainty propagation in analytic availability models, *Reliable Distributed Systems, 2010 29th IEEE Symposium on*, pp. 121–130 (2010).

[4] Eclipse. `http://www.eclipse.org/`.

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

[6] Eramo, R., Pierantonio, A. and Rosa, G.: Uncertainty in bidirectional transformations, *Proceedings of the 6th International Workshop on Modeling in Software Engineering*, pp. 37–42 (2014).

[7] Esfahani, N., Kouroshfar, E. and Malek, S.: Taming uncertainty in self-adaptive software, *Proceedings of the 19th ACM SIGSOFT symposium and the 13th Euro-*

pean conference on Foundations of software engineering, pp. 234–244 (2011).

[8] Esfahani, N., Malek, S. and Razavi, K.: GuideArch: guiding the exploration of architectural solution space under uncertainty, *Software Engineering (ICSE), 2013 35th International Conference on*, pp. 43–52 (2013).

[9] Esfahani, N. and Malek, S.: Uncertainty in self-adaptive software systems, *Software Engineering for Self-Adaptive Systems II*, pp. 214–238 (2013).

[10] Esfahani, N., Razavi, K. and Malek, S.: Dealing with uncertainty in early software architecture, *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, p. 21 (2012).

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

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

[13] Famelis, M., Salay, R., Di Sandro, A. and Chechik, M.: Transformation of Models Containing Uncertainty, *Model-Driven Engineering Languages and Systems*, pp. 673–689 (2013).

[14] Fukamachi, T., Ubayashi, N., Hosoai, S. and Kamei, Y.: Modularity for Uncertainty, *Proceedings of the 7th International Workshop on Modeling in Software Engineering*, pp. 7–12 (2015).

[15] Garlan, D.: Software engineering in an uncertain world, *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pp. 125–128 (2010).

[16] Git. `https://git-scm.com/`.

[17] JGit. `http://www.eclipse.org/jgit/`.

[18] Letier, E., Stefan, D. and Barr, E. T.: Uncertainty, risk, and information value in software requirements and architecture, *Proceedings of the 36th International Conference on Software Engineering*, pp. 883–894 (2014).

[19] Llerena, Y. R. S.: Dealing with uncertainty in verification of nondeterministic systems, *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 787–790 (2014).

[20] Magee, J. and Kramer, J.: *State Models and Java Programs*, Wiley (1999).

[21] Massey, A. K., Rutledge, R. L., Antón, A. I. and Swire, P. P.: Identifying and classifying ambiguity for regulatory requirements, *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, pp. 83–92 (2014).

[22] Meedeniya, I., Moser, I., Aleti, A. and Grunske, L.: Architecture-based reliability evaluation under uncertainty, *Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS*, pp. 85–94 (2011).

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

[24] Popstojanova, K. G. and Kamavaram, S.: Assessing uncertainty in reliability of component-based software systems, *Software reliability engineering, 2003. ISSRE 2003. 14th international symposium on*, pp. 307–320 (2003).

[25] Raccoon and Dog: Unknownness, *ACM SIGSOFT Software Engineering Notes*, Vol. 38, No. 5, pp. 8–17 (2013).

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

[27] Salay, R., Chechik, M., Horkoff, J. and Di Sandro, A.: Managing Requirements Uncertainty with Partial Models, *Requirements Engineering*, Vol. 18, No. 2, pp. 107–128 (2013).

[28] Salay, R., Gorzny, J. and Chechik, M.: Change Propagation Due to Uncertainty Change, *Fundamental Approaches to Software Engineering*, pp. 21–36 (2013).

[29] Sommerville, I.: Integrated requirements engineering: A tutorial, *Software, IEEE*, Vol. 22, No. 1, pp. 16–23 (2005).

[30] Trubiani, C., Meedeniya, I., Cortellessa, V., Aleti, A. and Grunske, L.: Model-based performance analysis of software architectures under uncertainty, *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, pp. 69–78 (2013).

[31] Ubayashi, N., Nomura, J. and Tamai, T.: Archface: A Contract Place Where Architectural Design and Code Meet Together, *Proceedings of the 32nd International Conference on Software Engineering*, pp. 75–84 (2010).

[32] Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. and Bruel, J.-M.: RELAX: a language to address uncertainty in self-adaptive systems requirement, *Requirements Engineering*, Vol. 15, No. 2, pp. 177–196 (2010).

[33] Xtext. `https://eclipse.org/Xtext/`.

[34] Yamashita, K., Jiang, G., Fukamachi, T., Kamei, Y. and Ubayashi, N.: An Empirical Study of Uncertainty in GIMP Project, *International Workshop on Empirical Software Engineering in Practice (IWESEP), Poster Session* (2016).

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

[36] Ziv, H., Richardson, D. and Klösch, R.: The uncertainty principle in software engineering, *submitted to Proceedings of the 19th International Conference on Software Engineering (ICSE'97)* (1997).

[37] 深町拓也，鵜林尚靖，細合晋太郎，亀井靖高：不確かさを包容する Java プログラミング環境，情報処理学会研究報告ソフトウェア工学研究会報告，Vol. 2015-SE-189, No. 21, pp. 1–8 (2015).

[38] 深町拓也，鵜林尚靖，細合晋太郎，亀井靖高：不確かさを包容するソフトウェア開発プロセス，ソフトウェア工学の基礎，Vol. 41, pp. 47–52 (2015).

[39] 艾迪，鵜林尚靖， 李沛源，李宇寧，細合晋太郎，亀井靖高：設計抽象化のためのリファクタリングパターン，情報処理学会研究報告ソフトウェア工学研究会報告，Vol. 2014-SE-185, No. 19, pp. 1–8 (2014).