Vol.2016-SE-191 No.3
2016/3/14

開発者の意図に沿ったデバッグ支援環境 DebugConcierge

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

概要：現在，様々な自動バグ修正技術が提案されている．しかしながら，適切な修正を行うのは，まだ現実的でない．そのため，開発者が修正を行うことの支援として，本稿では既存のバグ修正手法の利点を取り込み，情報推薦の高度化を図った，デバッグ支援環境 DebugConcierge を提案する．本ツールは，ローカル知識からの修正情報と，クラウド知識からの修正情報を組み合わせて，修正情報を推薦する．また，開発者のコンテキストを推測し，考慮することで，開発者が意図した修正を行えるようクラウド知識から情報を推薦する．デバッグ支援機能として，自動修正と修正状態管理の機能も提供する．この DebugConcierge を使って，オープンソースプロジェクトの Rhino のバグを，どう修正するのかを検証した．

1. はじめに

ソフトウェア工学において，ソフトウェア保守というのは重要な研究領域である．ソフトウェア保守は，既存のソフトウェアを改良・最適化していくことや，バグを修正していくことである．ソフトウェアからバグを取り除くことは，多くの場合，手作業である．手動でのバグ修正には多くの手間と時間がかかる．そのため，手動でのバグ修正を効率化することは意義のあることである．

バグ修正を効率的に行う支援の理想としては，自動でバグを修正してくれるものがあげられる．自動バグ修正の研究 [3][6][8][9] は多くなされている．しかしながら，これらの自動バグ修正の手法を用いて，適切な修正をするのは，まだ現実的でない [10]．一方で，自動バグ修正ツールは適切な修正を行うことは難しいが，暫定的な修正であれば施すことができる．これらの事から，現在のところ適切なバグの修正には，人間の判断が必要であると考えられる．

人間がバグを修正するのを支援する研究としては，バグの修正に関するヒントを作成する研究 [4][5] が行なわれている．これらの研究では修正ヒントを作成する際に，ソースコードとテストを利用したり，ウェブ上のクラウド知識を利用したりしている．先に述べた自動バグ修正の手法は，ソースコードとテストを利用している．自動バグ修正の手法により生成された修正コードも，バグ修正ヒントもバグ修正に関する情報であるが，これらの情報源は，ソースコードとテストというローカルな知識，ウェブ上のクラウド知識の 2 つに分けられる．これらローカル知識とクラウド知識から生成された修正情報は異なる特徴を持つと考えられる．そのため，どちらかの手法単体から得られる情報では，修正できるバグの種類に制限があると考えられる．

ウェブ上のクラウド知識を用いて，バグに関する情報を集める場合，クラウド知識の量は膨大であるため，開発者が満足する情報を得られないことが多い [11]．そのため，開発者が求める情報を開発者のコンテキストから推測する必要がある．本稿において，開発者のコンテキストとは，開発環境や使用言語，開発者がどういった修正を行いたいかという意図とする．

そこで本稿では，既存のバグ修正手法の利点を取り込み，情報推薦機能の高度化を図った，デバッグ支援環境 DebugConcierge を提案する．DebugConcierge は，自動バグ修正ツール Kali[10] のアイデアを利用した修正候補と Stack Overflow のクラウド知識ベースからの修正ヒントを推薦する．また，デバッグ支援機能として，自動修正機能と修正状態管理機能を提供する．

情報推薦では，開発者のコンテキストに沿ったクラウド知識上の情報の検索と，異なる特徴を持った修正情報の提示で，バグ修正の効率化を図る．自動修正では，提示された修正情報を自動でソースコードに反映することで，修正コードのタイピングの手間を省き，間違ったコードが混入することを防ぐ．修正状態管理では，短時間で出来る応急処置的な修正を暫定修正として管理し，この暫定修正が将来的にバグの原因とならないように管理する．

本稿では，2 章で，関連研究を挙げる．3 章で，修正情

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

報の特徴を述べる．4 章で，デバッグを支援するアイデアを述べる．5 章で，このアイデアを元に実装した DebugConcierge というデバッグ支援環境を提案する．6 章で，DebugConcierge を使ってバグ修正時の問題をどう解決するかを説明する．7 章で，まとめと今後の課題を述べる．

2. 関連研究

2.1 自動バグ修正

デバッグを支援する手段としては，自動バグ修正があげられる．自動バグ修正の研究は，多くなされている．GenProg[3] は，遺伝的アルゴリズムを用いて，コードの変更を行い，考慮される全てのテストケースをパスすること目指して，パッチを生成する．RSRepair[8] では，GenProg で遺伝的アルゴリズムを使っていた代わりに，ランダムサーチアルゴリズムを用いて，パッチを生成する．AE[9] は，決定的アルゴリズムを用いてパッチを生成し，等価なパッチを枝刈りするためにプログラムの等価関係を利用している．SemFix[6] は，記号的実行と制約解決を用いて，バグのある部分を置き換えて，バグの修正を行っている．Angelic Debugging[2] は，記号的実行を利用して，今までパスしていたテストに対してパスする状態を維持しつつ，失敗したテストをパスできるようにコードに変更を加えていく手法を提案している．Pei ら [7] は，ソースコードの静的解析と，テストを用いた動的解析の 2 つの解析を行ってバグの修正を試みている．

自動バグ修正は，手作業でバグ修正を行うのに比べると，魅力的ではあるが，現在のところ制限もある．1 つ目に，テストなど仕様に頼っていることがあげられる．自動バグ修正によって作成された，修正の候補が本当に正しいのか判断する材料としてテストケースが用いられている．これにより，修正候補が，テストケースはパスするが，適切な修正でないという問題が起こる．2 つ目に，修正に時間がかかることがあげられる．自動バグ修正では，全てのテストをパスするような修正候補を探すため，解となる修正コードがあるかもしれない場所を探索しなければならない．その探索空間が，適切な修正を求めるあまり，大きくなってしまうため時間がかかってしまう．

また，Qi ら [10] によると GenProg，RSRepair，AE による修正の多くは，機能を削除するような修正であり，適切な修正ができていないと言われている [10]．また，Qi らは，機能を削除するような修正だけを行う Kali という自動バグ修正ツールを作成し，機能を削除するような修正だけで修正効果を検証した．結果として GenProg，RSRepair，AE と同じくらいの効果が得られた．

このため，現在のところ自動バグ修正だけで適切なデバッグを行うのは難しいと考えられる．ただし，機能の一部を使う形でなら役立てることは可能である．

自動バグ修正だけでデバッグ支援として行うのは現在難しい [10] ので，デバッグ作業を細分化して，各作業を支援することで総合的にデバッグの支援を行う．

2.2 ヒントの作成

開発者がパッチを作成するのを支援する研究としては，修正ヒントを与える研究がある．

MintHint[5] は，記号的実行を使って，テストケースから修正の候補を推論し，その候補を相関値でランク付けして，修正ヒントとして開発者に提示する．Crowd Debugging[4] では，クラウド知識を利用して修正ヒントを生成している．Crowd Debugging は，Stack Overflow[*1] というプログラミングに特化した Q&A サイトの情報を基にしてバグ修正のためのヒントを作成する．Stack Overflow には，他の Q&A サイトに比べて，多くのコード片が含まれており，ヒントとして参考にしやすい．Crowd Debugging は，Stack Overflow 中の質問者側の投稿に含まれているコード片をバグの候補として，ソースコード中を探査し，類似したコードを見つけ，開発者に提示する．

3. 修正情報の特徴

3.1 ソースコードとテストによるバグ修正情報

ソースコードとテストによるバグ修正情報には適切なバグ修正を行えないものがある．GenProg[3] や MintHint[5] などのソースコードとテストから生成されるバグ修正情報は，修正情報の生成の過程でテストを利用している．GenProg は生成したパッチの妥当性を測るために，MintHint ではヒントを生成するための資源としてテストを用いている．これらは，テストによりソフトウェアの振る舞いが規定されるという前提に依存している．これは換言すると，テストさえ通れば，その修正情報は適切であると言えるということである．しかし，テストが網羅し損ねている部分に関して不安がある．テストにはパスするが適切でない修正ということが発生しうる．そのため，バグ修正の過程で人間の判断を挟む必要がある．

また，ソースコードとテストによるバグ修正情報は，どんなバグでも修正できるわけではないと考えられる．GenProg や MintHint などが修正情報の候補を探す空間がソースコード中であるということがその原因である．GenProg では，コードを変異させる際にはコード片を埋め込むことがある．そのコード片の候補がソースコード中から選ばれる．MintHint ではテストをパスするような条件を満たすようなコード片がソースコード中から選ばれる．このことから，適切な修正となるようなコードがソースコード中にないような場合，適切な修正が行われないと考えられる．

[*1] http://stackoverflow.com/

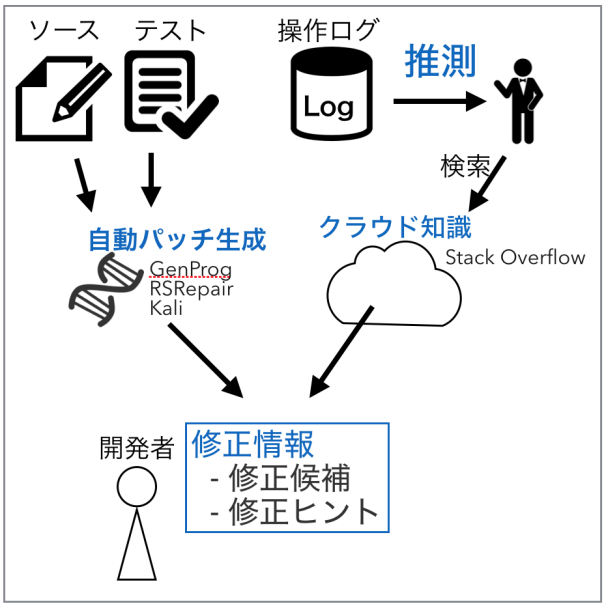

図 1 情報推薦アイデア

3.2 クラウド知識によるバグ修正情報

クラウド知識によるバグ修正情報は，発生頻度の高いバグに対してはバグ修正に有益な情報をもたらすが，発生頻度が低いバグに対しては情報をもたらせないことがある．Crowd Debugging[4] では，クラウド知識として Stack Overflow からバグ修正情報を生成する．Stack Overflow には，問題を抱えた開発者が質問を投稿し，その質問に答えるような質疑応答の情報が多く蓄えられている．Stack Overflow には多くの重複した内容の投稿があることが報告されている [1][4]．重複するような投稿は，多くの開発者が直面するものであると考えられる．そのため，発生頻度の高いバグに関する情報は多くもたらされ，発生頻度が低いものに関する情報は少なくなる．

4. デバッグ支援

4.1 情報推薦での支援

バグを修正する際に，開発者に有益な情報を情報を提示するには，まず開発者が解決したいことを予測する．本稿では，開発環境，使用言語，開発者がどういう修正を行いたいかをコンテキストと表すことにする．コンテキストを予測した後，これについての修正情報を集める．

4.1.1 開発者の行動を元にコンテキストを推測

開発者のコンテキストを予測するために開発環境の操作ログを使用する．例えば，ある同じファイルを編集しては，コンパイルと実行を繰り返すようなら，そのファイルにバグが存在すると判断できる．よって，そのファイル中の情報をもとに修正情報を収集する．これにより，開発者が求める修正情報を正確に提示することができ，また，全てのファイルを対象とした修正情報の探索を行わなくて良いので，修正情報の生成に時間がかかりにくい．

同時に，開発者はデバッグ中であるとわかっており，適切なタイミングでの情報推薦により，今までは，バグ修正時に明示的に情報の検索を行わなければならなかった手間を省くことができる．**図 1** にイメージを示す．図中の開発者が使用している開発環境には，エディタと開発環境上での操作のログが残っている．このログを基にして，開発者が今行っている作業を推測する．推測の結果を基に，クラウド知識から情報を取り出して，開発者に提供する．

4.1.2 有益な情報の提供

有益な情報を提供するには，異なる特徴を持つ情報を組み合わせることが重要だと考えられる．3.1 節で述べたように，ソースコードとテストによる修正情報は，テストをパスするが適切でない修正を行う可能性があり，適切な修正をできるバグの種類にも制限がある．また，3.2 節で述べたように，クラウド知識からの修正情報は，バグに関する修正情報の有益さがそのバグの発生頻度による．どちらの修正情報も単体では，有益さに限界がある．そこで，両方の修正情報を開発者に提示し，その情報をどう扱うかは開発者の判断で決めてもらう．どちらの情報も本質的な修正の役に立たなくても，ソースコードとテストによる修正情報は，その生成の過程からわかるように，テストをパスすることはできるので，応急処置的な修正として利用可能である．

4.2 自動修正での支援

ツールから提示される修正情報をそのままソースコードに反映し自動で修正できれば，開発者は修正情報を見て修正コードをタイプする手間を省くことができる．そこで，4.1.2 節で述べた情報推薦で提供された修正情報をソースコードに自動で反映させる．修正情報が自動バグ修正ツールから生成されたものであれば，パッチの形をしているので，そのままソースコードに反映することができる．一方，ヒント生成ツールから生成された修正情報の場合，修正ヒントというドキュメントの形であり，そのままではソースコードに反映することができない．そのため，まず，このヒントからコード部分を抜き出す．このコードとソースコードとを比較して，類似する部分を，変数名などはソースコードに合わせて埋め込んでいく．

自動修正を行った際に，反映されたコードにはトレーサビリティ確保のため注釈を付ける．なぜなら，自動修正を行った部分は全て暫定的な修正である可能性があるからである．本質的な修正であると開発者が判断すれば，この注釈を外す．

GenProg などで，適切なバグ修正は難しかった要因の一つには，テスト結果のみで修正コードが判断していたことが考えられる．そのため，このアイデアでは，修正コードをソースに反映させるかどうかは，開発者が判断するため，適切なバグ修正が望めると考えられる．この自動修正

表 1 関連研究と本研究の差異

	アプローチ	問題点	本研究でのアプローチ
GenProg[3]	遺伝的アルゴリズムによるパッチ生成	適切なパッチでない	—
RSRepair[8]	ランダムサーチによるパッチ生成	適切なパッチでない	—
AE[9]	決定性アルゴリズムによるパッチ生成	適切なパッチでない	—
Kali[10]	機能の削除によるパッチ生成	適切なパッチでない	暫定修正用の修正候補として採用
MintHint[5]	記号的実行によるバグ修正ヒント生成	—	—
Crowd Debugging[4]	クラウド知識による修正ヒント生成	低頻度のバグに効果が薄い	修正ヒントとして採用

表 2 返り値の対応

	void	int long	float double	他のクラス Class
T の中身	—	-1	-1.0	new Class()

により，プログラマが修正コードを記述する手間を省き，間違ったコードの混入を防ぐことができる．

4.3 修正状態管理での支援

暫定的な修正を施された部分を管理すれば，修正状態のトレーサビリティを確保でき，暫定修正したコードがバグの原因になるのを防ぐことができる．

まず，4.2 節で述べた自動修正を行ったときに付ける注釈を，開発者が手動でもつけられるようにする．なぜなら，開発者がバグ修正に十分な時間が取れずに，短時間で応急処置的なコードを記述することもあるからである．注釈をつけることで，後から応急処置であることを判別することができる．これで，時間的な課題を解決することができる．

コンパイル時に，注釈が付けられている部分を全て Warning として表示し，開発者に暫定的な修正箇所を知らせる．それにより，コンパイルの都度，開発者は本質的な修正を促されるので，暫定的な修正を施したことを忘れなくなる．これにより，暫定的な修正が開発者の認識の外に出ることがなくなり，暫定的な修正が将来のバグの原因になることを防ぐことができる．これにより，バグの再発に関する課題を解決する．

注釈を付ける際に，タグを付与し，どのバグを修正した際に変更されたコードなのかを識別する．これにより，修正箇所をタグで整理し，管理することができる．修正箇所が複数のファイルに渡ったとしても，タグにより追跡が可能なので，あるバグに関する暫定的な修正を本質的な修正に変更するときに変更漏れが発生しなくなる．これにより，横断的関心事に関する課題を解決することができる．

コードの状態を管理することに類似して，バージョン管理システム git などで修正状態を管理が考えられる．しかし，git だと管理単位の粒度が大きく，修正状態管理に向かない．git を拡張するなどして，もう少し小さな粒度で，例えば，ソースコードの 1 ステートメント単位で管理する必要がある．

5. DebugConcierge 概要

本稿で提案するデバッグ支援環境 DebugConcierge は，Eclipse のプラグインとして実装した．対象とするプログラミング言語は Java である．DebugConcierge は，コード修正案の推薦，コードの自動修正，修正レベルの管理の 3 つの機能を提供する．

5.1 今回の実装範囲

プロトタイプとして，一部の機能を実装した．**表 3** に全体機能を示す．図を参照すると，情報推薦全体は，自動バグ修正ツールとの連携と，開発者のコンテキストを推測，CodeConcierge[11] との連携からなる．今回実装したのは，自動バグ修正ツールとの連携と，CodeConcierge との連携である．ただし，今回，自動バグ修正ツールとの連携に関しては，ツールの処理速度に不安があるため，Kali[10] のアルゴリズムを参考した実装を行った．これらのプロトタイプ実装を優先した理由としては，デバッグ支援機能が推薦された情報を利用することを前提としているので，修正情報を推薦する必要があったためである．デバッグ支援機能に関しては，機能全体として，自動修正機能と修正状態管理機能がある．今回は，自動修正機能に関しては，修正候補を反映を実装した．修正候補の反映を優先した理由としては，自動バグ修正ツールが生成した修正コードを反映すればよく，実装に時間がかからないためである．修正状態管理機能が自動修正機能に依存しているため，自動修正機能を駆動するようにする必要があった．修正状態の管理機能に関しては，基本的な機能を実装している．

5.2 修正情報の推薦

DebugConcierge では，修正候補と修正ヒントの 2 種類の修正情報を推薦する．修正候補は，Kali[10] からのアイデアを利用した修正情報である．Kali は，機能を削除するような修正だけで自動バグ修正を試みるツールである．機能を削除するような修正とは，具体的には，return 分の挿入、条件分岐の固定化，コードブロックの削除である．return 文の挿入については，return 文をエディタのカーソルの位置に return 文を挿入するような内容の修正情報である．条件文の固定化については，ソースコード中の if 文

表 3 実装の範囲

機能	プロセス	具体的な処理	プロトタイプ
情報推薦機能	修正候補の生成	自動バグ修正ツールとの連携	Kali[10] を参考
	修正ヒントの生成	開発者のコンテキストの推測	—
		CodeConcierge[11] との連携	実装完了
デバッグ支援機能	自動修正	修正候補の反映	実装完了
		修正ヒントの反映	—
	修正状態管理	暫定修正の管理	実装完了

を検索し，その if 文の条件式部分を true に置換するような内容の修正情報である．コードブロックの削除では，エディタ中の選択部分をコメントアウトし，コメントアウトするような内容の修正情報である．

修正ヒントは，StackOverflow[*2] のクラウド知識による修正ヒントである．開発者は，キーワードか，ソースコードの一部をもとにして，クラウド知識ベースからバグ除去の参考となる情報（修正例や API ドキュメントなど）を検索することができる．この情報検索機能には，プログラミング支援ツール CodeConcierge[11] の API を利用する．入力されたキーワードとエディタ中で選択状態のコード片を HTTP リクエストで CodeConcierge に送信する．それから，json 形式で検索結果を受け取り，解析，整形して開発者に提供する．開発者は提供された修正ヒントをダブルクリックすることでブラウザで StackOverflow の該当ページを開いて修正情報を得ることができる．

5.3 コードの自動修正

プログラマが選択した修正情報を実際にコードに反映させていく．DebugConcierge は，情報推薦で提示した修正候補に対して，自動修正が行う機能を提供する．修正候補をダブルックリックすると，修正候補に応じてソースコードの自動修正を行う．return 文の挿入では，エディタ上のカーソルがある位置に return <T>; を挿入し，挿入されたコードに注釈を付ける．<T >は，挿入されるメソッドの返り値に応じて変更される．**表 2** に，<T>の部分がどう変更されるか示す．条件文の固定化では，if 文の条件式部分を true に置換し，置換された部分に注釈を付ける．コードブロックの削除では，エディタ中の選択部分をコメントアウトし，コメントアウトされた部分に注釈を付ける．

5.4 修正状態の管理

暫定的な修正によるコードの変更を，プログラマが後から追跡できるよう，行われた修正の状態を管理する．暫定的な修正だけに注釈をつけ，何も注釈が付けられていないものは本質的な修正として扱う．このように修正状態を分けて，修正の状態を管理する．エディタ上に注釈を表示する Eclipse のフレームワークを用いて，暫定的な修正を行った箇所をハイライトする．注釈を付けるには，5.3 節で述べた自動修正を行うか，注釈をつけるメニューを選択する方法がある．自動修正では，埋め込まれたコードに注釈が付く．注釈をつけるメニューでは，選択状態であるコードに注釈が付けられる．注釈をつける際には，注釈に関連づけるタグを管理するマネージャが出現する．このマネージャで，新規にタグを登録し関連づけするか，既にあるタグを関連づけるか決めて注釈を作成する．暫定修正管理ビューでは，暫定修正の一覧とその修正に関連付けれれているタグを確認することができる．

また，注釈と連携して，クイックフィックスを提供する Eclipse のフレームワークを用いて，注釈が付いてる部分を削除したり，注釈だけを削除したりするクイックフィックスを実装した．これにより，注釈にマウスカーソルを合わせることで，これらのクイックフィックスを使用することができる．

6. ケーススタディ

6.1 今回用いるプログラム例の概要

Rhino[*3] を用いて，DebugConcierge の説明を行う．Rhino は，Java で記述された，オープンソースで開発されている JavaScript の実装である．Rhino のバグ#217379 を修正することを仮定して，DebugConcierge の使用法を説明する．このバグは，String.prototype.replace(re, func) というメソッドで NullPointerException が発生するというものである．前提として，バグの位置は何らかの手法を用いて，既に特定されているとする．**図 3** に String.prototype.replace(re, func) の実装コードの一部で，バグの箇所付近のコードを示す．図 3 中の 7 行目で NullPointerException が発生している．

6.2 DebugConcierge を用いたデバッグ

図 2 にユーザーインタフェースの外観を示す．図中①は，実際にコードタイプするエディタである．自動修正は，ここに表示されているコードに対して行われる．図中②は，情報推薦用のビューである．ビュー右上のテキストボックスにキーワードを入力するか，①中でコードの一部を選択状態にした状態で，Search ボタンを押せば，ソースの解析

[*2] http://stackoverflow.com/

[*3] https://developer.mozilla.org/ja/docs/Rhino

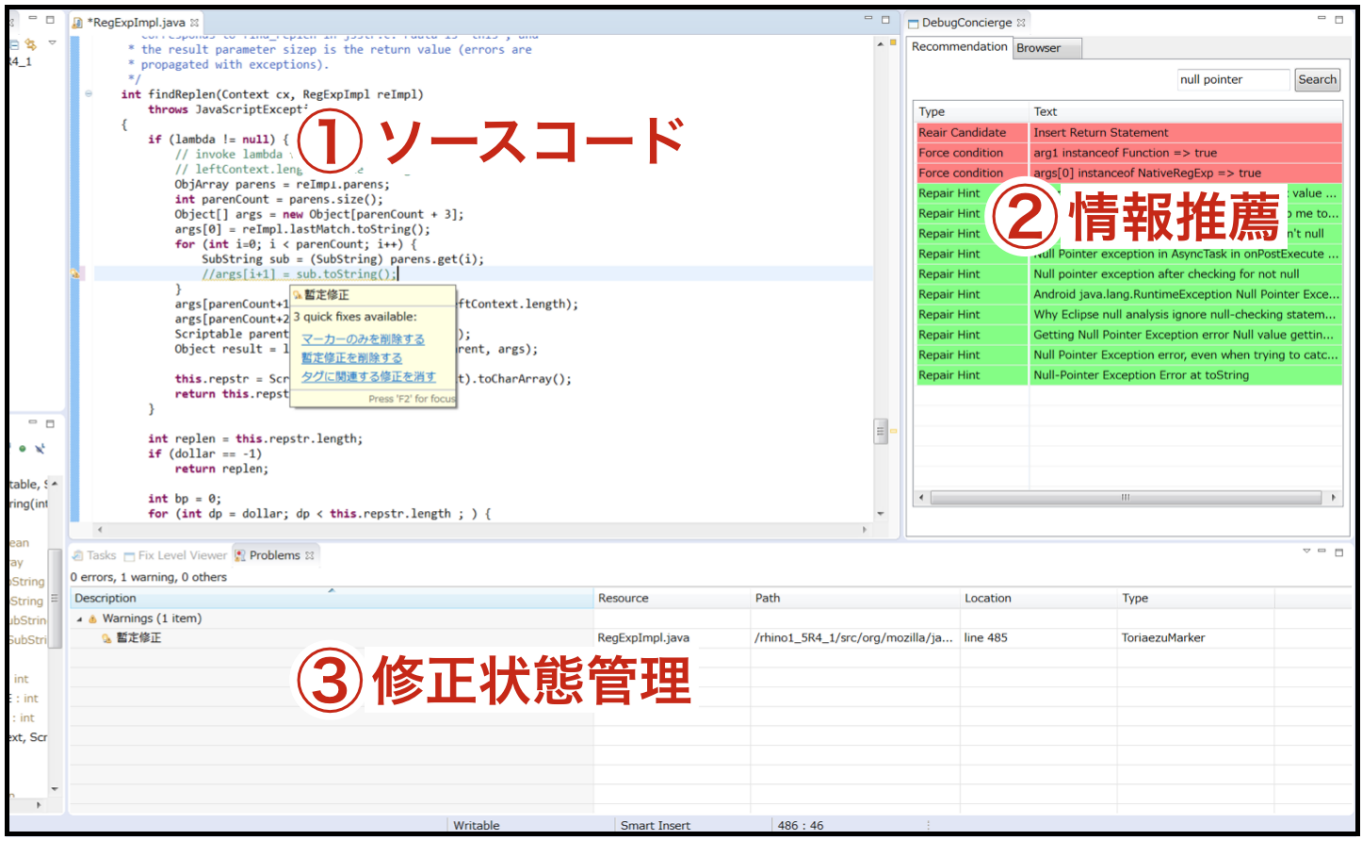

図 2 DebugConcierge の概観

```
1  ObjArray parens = reImpl.parens;
2  int parenCount = parens.size();
3  Object[] args = new Object[parenCount + 3];
4  args[0] = reImpl.lastMatch.toString();
5  for (int i=0; i < parenCount; i++) {
6          SubString sub = (SubString) parens.get(i);
7          args[i+1] = sub.toString();
8  }
9  args[parenCount+1] = new Integer(reImpl.leftContext
       .length);
10 args[parenCount+2] = str;
11 Scriptable parent = lambda.getParentScope();
12 Object result = lambda.call(cx, parent, parent, args);
13
14 this.repstr = ScriptRuntime.toString(result).
       toCharArray();
```

図 3 バグ#217379

器が実行され Kali で実装されていた機能を削除するような修正候補を提示するとともに，クラウド知識ベースから修正情報を得ることができる．図中③は，修正状態の管理を行うためのビューである．暫定的な修正を行ったバグの箇所やそれに付随するタグの情報の一覧が表示される．

6.2.1 修正情報の収集

修正情報を集めるために**図 4** のように，修正箇所を選択状態にし，不具合で NullPointerException が出るので，null や pointer といったキーワードをテキストボックスに入力する．それから Search ボタンを押すと，図 4 にあるように，Repair Candidate や Force Condition のような修正候補と Repair Hint といった修正ヒントが提示される．

6.2.2 バグの修正

修正ヒントをダブルクリックすれば，**図 5** のようにタブを開いて，修正ヒントとなるページを見ることができる．このページの情報をもとにバグを解決する．自動バグ修正の修正候補は，現在のところ，限定的な条件下での応急処置にしか使えない．図 5 の Stack Overflow の情報によると sub が null になっているため NullPointerException が発生している．そこで sub が null か調べてから，args[i+1] = sub.toString(); を実行するよう，if(sub != null) args[i+1] = sub.toString(); と変更する．

6.2.3 修正状態の登録

修正が行われたら，次に修正状態の管理に移行する．今回の修正は，将来的に修正の方法が変わるかもしれない暫定的な修正であるので，暫定的な修正であることを示す注釈を付ける必要がある．**図 6** のポップアップメニューから修正箇所に対して注釈をつけていく．修正箇所を選択状態にして，ポップアップメニューの暫定修正マーカー設置を選択すると，タグマネージャのダイアログが開く．このタグマネージャーにタグを新規に追加し，この注釈と関連づける．これらのマーカーが付けられた部分は，コードエ

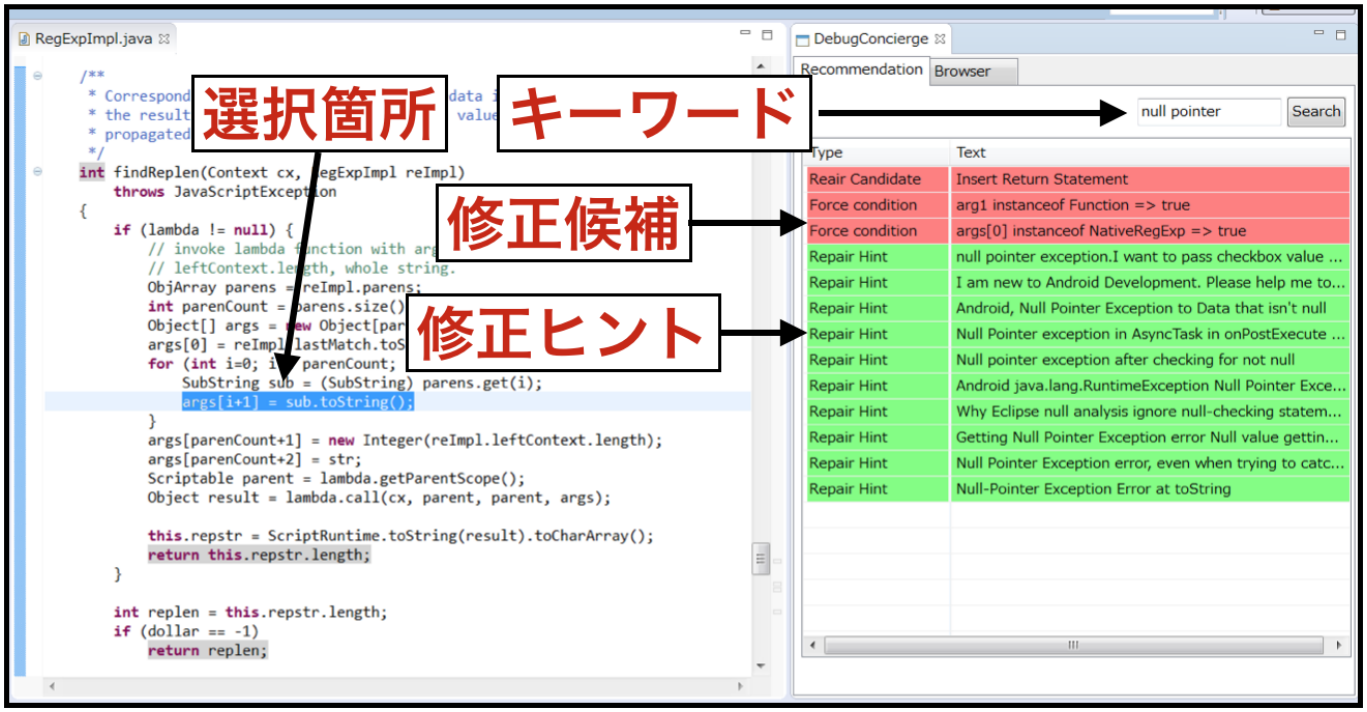

図 4 情報取集時の画面

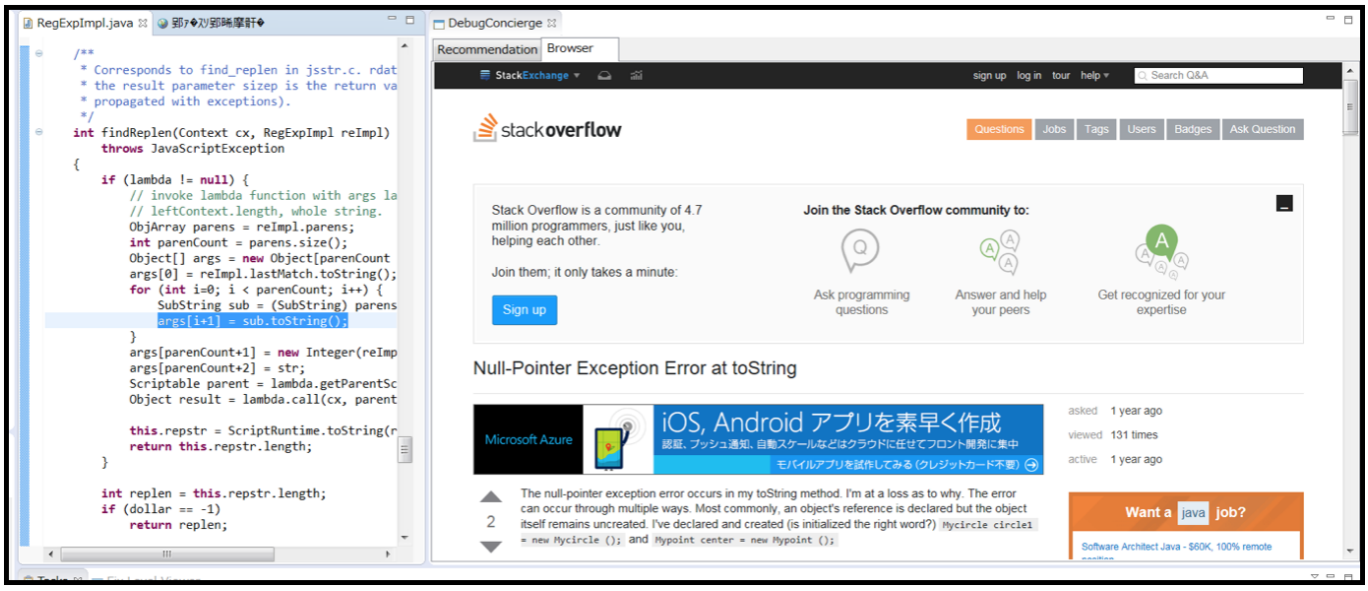

図 5 修正ヒント参照時の画面

ディタ上で注釈がつき，目立つようになる．さらに，**図 7** のように，コンパイル時に Warning として表示され，暫定的な修正の箇所を忘れにくくしている．

6.3 考察と今後の課題

まず，修正候補について，Kali のアイデアを利用しているので暫定的な修正が可能であるかもしれないが，自動バグ修正ツールと連携したもので無いので，暫定的な修正ができることが保証されていない．修正ヒントについては，今回のケーススタディから分かるように，参考になるヒントが順序の低いところに現れている．他の修正ヒントを参照すると，今回 Java のプロジェクトを対象にしたが，全 10 件のヒントの内，3 件が Android に関するものであり，対象とする開発環境，フレームワークをコンテキストとして含めることができれば，もっと有用な情報を絞り込めると考えられる．また，ほとんどのヒントが null pointer というキーワードに関連づいたものである．これは，今回コンテキストとして含めたソースコード片の特徴が弱かったためと考えらる．ソースコード片中の toString というメソッド名に関連があるものが 1 件推薦されている．この

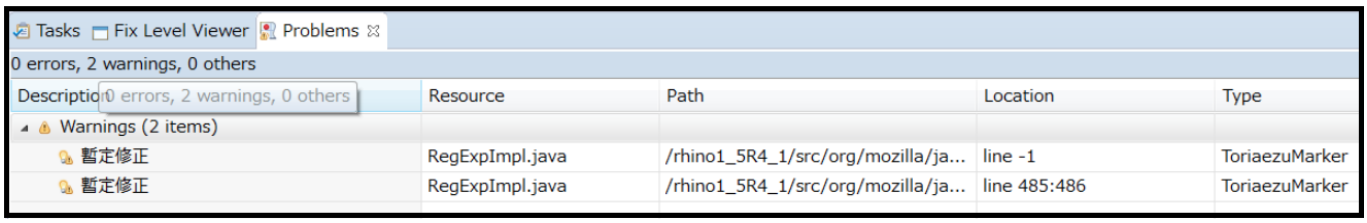

図 7 Warning 表示

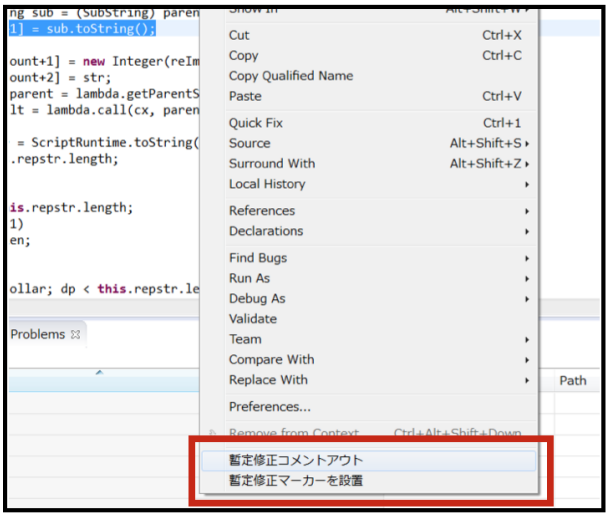

図 6 ポップアップメニュー

toString は Java の一般的なクラスのメソッドであるが，このことから，Java でよく使用されているクラスのメソッドに関するバグであれば，適切な情報が得られると考えられる．このことから，開発者のコンテキストとしてキーワードとソースコード片だけでは，少ないと考えられる．もっと多くのコンテキストを利用して，開発者にとって有用な情報を絞り込む必要がある．また，今回利用したコンテキストは，開発者が直接入手しているので，ツールが自動で推測する必要がある．

7. まとめ

本稿では，既存のデバッグ手法の利点を取り込み，バグ修正に関する情報推薦の高度化を図ったデバッグ支援環境 DebugConcierge を提案した．DebugConcierge は，ソースコードとテストから生成された修正情報と，開発者のコンテキストを考慮したクラウド知識からの修正情報を組み合わせて，修正情報を推薦する．また，この情報推薦機能の支援として，自動修正と修正状態管理の機能も提供する．ケーススタディとして，DebugConcierge を使って，どうバグを修正するのかを説明した．

参考文献

[1] Allamanis, M. and Sutton, C.: Mining Idioms from Source Code, *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, New York, NY, USA, ACM, pp. 472–483 (online), DOI: 10.1145/2635868.2635901 (2014).

[2] Chandra, S., Torlak, E., Barman, S. and Bodik, R.: Angelic Debugging, *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pp. 121–130 (online), DOI: 10.1145/1985793.1985811 (2011).

[3] Claire, Le, G., ThanhVu, N., Stephanie, F. and Westley, W.: GenProg: A Generic Method for Automatic Software Repair, *Software Engineering, IEEE Transactions on*, Vol. 38, No. 1, pp. 54–72 (online), DOI: 10.1109/TSE.2011.104 (2012).

[4] Fuxing, C. and Shunghun, K.: Crowd Debugging, *Proceeding of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering(ESEC/FSE)*, pp. 320–332 (2015).

[5] Kaleeswaran, S., Tulsian, V., Kanade, A. and Orso, A.: MintHint: Automated Synthesis of Repair Hints, *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, New York, NY, USA, ACM, pp. 266–276 (online), DOI: 10.1145/2568225.2568258 (2014).

[6] Nguyen, H. D. T., Qi, D., Roychoudhury, A. and Chandra, S.: SemFix: Program Repair via Semantic Analysis, *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, Piscataway, NJ, USA, IEEE Press, pp. 772–781 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2486788.2486890⟩ (2013).

[7] Pei, Y., Wei, Y., Furia, C. A., Nordio, M. and Meyer, B.: Code-based Automated Program Fixing, *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, Washington, DC, USA, IEEE Computer Society, pp. 392–395 (online), DOI: 10.1109/ASE.2011.6100080 (2011).

[8] Qi, Y., Mao, X., Lei, Y., Dai, Z. and Wang, C.: The Strength of Random Search on Automated Program Repair, *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, New York, NY, USA, ACM, pp. 254–265 (online), DOI: 10.1145/2568225.2568254 (2014).

[9] W., W., Z., P. F. and S., F.: Leveraging program equivalence for adaptive program repair: Models and first results, *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pp. 356 – 366 (2013).

[10] Zichao, Q., Fan, L., Sara, A. and Martin, R.: An Analysis of Patch Plausibility and Correctness for Generate-And-Vlidate Patch Generation Systems, *Proceedings of the 2015 International Symposium on Software Testing and Analysis(ISSTA'15)*, pp. 24–36 (2015).

[11] 高橋裕太，鵜林尚靖，細合晋太郎，亀井靖高，渡邉卓也：Code Concierge: クラウド知識に基づいたプログラミング支援，ウィンタワークショップ 2016・イン・逗子，pp. 14–15 (2016).