2008－SE－159 (3)
2008／3／17

# 要求追加によるインパクトの分析に基づく組込みソフトウェア開発の効率化

川平航介†, 長田晃†, 海谷治彦†, 北澤直幸†, 海尻賢二†

†信州大学大学院

要旨

近年，組み込みソフトウェアは急速に大規模化・複雑化を続けている上に，開発期間の短縮が求められている．そのためは，既存ソフトウェアの再利用が必須となるが，実際の開発では開発に関する種々のドキュメントやソースコードの管理が不十分である場合も多い．既存のソフトウェアを再利用して開発を開始する場合に，開発者への負担を最小限に抑えつつ，要求の変更によるソフトウェアの修正箇所を明らかにすることが重要である．本稿では，既存の組込みシステムへの要求追加により，修正しなければならないソースコードの箇所を，ソースコード中の語句から半自動的に特定する方法を実際の開発データに基づき模索した結果を報告する．

## Improving the efficiency of embedded software developments based on impact analysis by requirements change

Kousuke Kawahira†, Akira Osada†, Haruhiko Kaiya†, Naoyuki Kitazawa†, Kenji Kaijiri†

†Graduate School of Science and Technology, Shinshu University

Abstruct

The size and complexity of software in embedded systems are rapidly increasing, but the development schedule is shortened due to the market pressure everyday. Reuse of existing software artifacts, e.g., design documents and source codes, is one of the promising ways to resolve this difficulty, but there are not enough artifacts suitable for such reuse because of the insufficient management of such artifacts. In this paper, we report an empirical study to investigate what kinds of reuse can be achieved under a real embedded software development. We mainly focus on impacts on source codes caused by requirements change because finding such impacts is required for efficient software development reusing existing software artifacts. We applied semi-automated impact analysis based on words appearing in source codes, and most impacts in source codes can be detected without explicit traceability links.

## 1．はじめに

組み込みソフトウェアは至る所で利用され，我々の日常生活において無くてはならない存在となっている．さらに近年，組込みソフトウェアは急速に大規模化・複雑化し続けており，開発のための工数も増大している．その上，開発期間の短縮や高品質化，低コスト化が求められているのが現状である．中でも，携帯電話やデジタルカメラ，プリンタなどの民生用機器では市場のニーズに合わせて次々に新製品を投入しなければならないため，品質を下げずに開発期間を短縮することが急務である．

大規模で高品質なソフトウェアを短納期・低コストで開発するためには既存ソフトウェアの再利用が不可欠である[1,2]．実際，新しい製品を開発する場合に，何も無いところから開発を始めることは少なく，多くのソフトウェア部品を再利用したり，既存のソフトウェアに変更を

加えて新しいソフトウェアを開発することがほとんどである．

効率よくソフトウェアに変更を加えるためには，仕様書とソースコードのトレーサビリティが重要であるが，実際の開発では不完全である場合も多い．また，開発の初期には，仕様書とソースコードの対応付けが十分になされている場合でも，開発が進むと仕様書とソースコードの乖離が発生しがちである．特に，組み込みシステムでは，メモリや処理速度の制約からオブジェクト指向で開発できない場合も多く，ある1つの機能が多くのモジュールに分散して実装されている場合も多い．また，ハードウェアとの同時開発の場合，ハードウェアの仕様変更がソフトウェアの仕様にも影響を及ぼす．そのため，仕様書とソースコードの対応を開発者が手作業で管理することは，大きな負担となる．このような理由が，組み込みソフトウェアの分野は他のソフトウェアに比べて，工学的な手法を取り入れにくくしている原因であると考えられる．

ソフトウェアの開発プロセスを改善する手法が提案され[3]，UMLを用いて組込みソフトウェアの開発を記述することも盛んになってきている[4]が，新たな開発手法を実際の開発に導入する場合，移行コストや開発者への負担が大きいなど，現場に浸透しない可能性が高い．そのため，現在持っている限られた仕様書やソースコード等のリソースを使って，開発を効率化する手法が必要である．

本論文では，組込みシステムの変更に対処するために仕様書とソースコードの対応付けをサポートする手法を提案する．実製品の２つのバージョンのデータを用いて分析を行い，旧バージョンのソフトウェアに対して要求が追加された場合に，ソースコード上の変更が必要な箇所を予測することを目指す．入力として旧バージョンのソースコード，変更要求の内容，新旧バージョンのハードウェア仕様書を用いてソースコードの修正箇所を予測する．

前もって，分析対象となるソフトウェアの２つのバージョンの間にある変更項目に注目し，実際に行われた変更を洗い出しておく．次に旧バージョンのデータから予測した変更箇所候補と比較する．この結果と，実際に行われた変更とを比較し，どの程度の精度と再現率で予測できるかを評価する．

２章で関連する研究について述べる．３章で適用する手法について延べ，４章で実際に分析を行う．５章で考察と課題をまとめる．

## ２．関連研究

ソフトウェアの動作実行をせずに非機能要求も含めてインパクトを分析するには，キーワードを用いた手法がが適している[5]．ソースコードと仕様書を現れるキーワードによって，自動的に関連付けを行う手法や，関連を管理して開発を助ける手法は現在までに，いくつか提案されている．

識別子を用いた仕様書とソースコードの関連付け手法としては，文脈に基づいたソースプログラムとドキュメント間の識別子対応付け手法[6]などがある．この手法ではソースコード上に現れる識別子に関して「宣言」と「参照」の区別をして，仕様書においても「定義」と「参照」に分けることで，ソースコードと仕様書の対応付けに役立てている．

また，組み込みソフトウェアのモジュール間の相互関係を可視化する管理ツールを使い効果的な再利用を実現する手法[7]も提案されている．この手法では，ソースコードやバージョン管理情報等，既存の資源を最大限に利用し可視化することで，開発者がソフトウェアの構造を理解する際に役立てている．労力やコストの追加を最小限に抑えた上で，再利用の効率化を図っている．

ソースコードとドキュメントの対応付けを管理する手法としてADIOS[8,9]が提案されている．ADIOSはアスペクト指向に基づきソースコードにドキュメントとの関連付けを埋め込む．関連を埋め込む作業は自動化されていないが，この手法を実現するツールであるADIOSによって埋め込まれた関連を収集し管理することができる．また関連に付加されたキーワードや種類の情報

により，関連を整理し種類やキーワードごとにまとめて扱うことができる．

## 3．キーワードによる変更予測手法

### 3.1 提案手法

変更予測を行う手法の概要を図１に示す．ソースコード上の，ある機能に関係している部分では，機能に関連の深い特定の変数や関数などの識別子が使われている可能性が高い．変更要求によって追加・変更された機能に関する仕様書の記述から，ソースコード上の識別子として使われる可能性が高いキーワードを抽出し，ソースコードから検索することで変更の可能性が高い箇所を予測する．その際，検索結果に現れる関数を見ながら最適と思われる結果が得られるように検索条件を調節する．

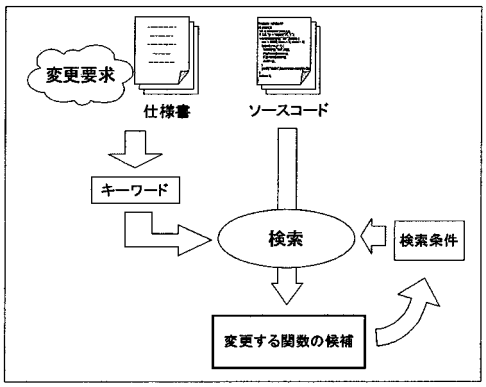

図１提案手法の流れ

仕様書に含まれるレジスタ名や特定の定数に付けられた名前は，ソフトウェアでも識別子として用いられる可能性が高い．今回用いた仕様書はハードウェア仕様書であるため，ハードウェアの視点で書かかれている．そのため，実際のソフトウェアに現れる識別子と完全に一致しないものも多いが，使われる単語はかなり共通していると考えられる．また，識別子を一度，単語単位に分解して用いることで，一致する可能性を高めることができる．それでも，仕様書から得られる語句のうち有用な語句の割合は限られているため，キーワードの抽出は開発者が手作業で行い必要に応じて語形などを変更することにした．

また，モジュール化されていない機能は色々な箇所に分散して実装されているが，その機能が働く条件が書かれていることが多い．そのため，機能に関連が深いキーワードは条件文等に現れることが多いと考えられる．キーワードの検索対象を条件文に絞ることにより，効率良く変更が必要な箇所を予測できる可能性があり，その観点についても分析を行う．

複数のキーワードとそれらの出現回数や出現する場所を入力として与えることにより，変更される可能性がある関数の一覧を出力するプログラムを作成し，分析に使用した．ソースコードを解析するツールとしてはSapid[10]などがあるが，今回は検索条件が単純なため，Perlでスクリプトを作成し，分析に用いた．また，予測結果の評価のために検索によって得られた関数のファイル中の位置や，実際に行われた変更を図示するプログラムも作成した．

### 3.2 入力データ

以下のものを変更予測の入力データとして用いる．

- 変更要求の内容
- ソースコード(旧)
- ハードウェア仕様書(新，旧)
- 既存ソフトウェアに関する知識

後ろに付加した「旧」「新」の文字は旧バージョン，新バージョンのものを使うことを示している．

本来，ソースコードとの対応を見つけるための仕様書はソフトウェアの設計書を用いたほうが，より多くの共通する語句を収集できる．しかし，今回の分析対象とした開発データでは，開発開始時にソフトウェアの詳細設計書が存在していなかったため，代替のリソースとしてそのソフトウェアが制御するハードウェアの仕様書を用いることとした．

### 3.3 手順

以下のような手順で変更を予測する．

1. 検索に使う単語の抽出
2. ソースコードから検索
3. 閾値と検索条件の調整

まず，検索に使用する単語の抽出を行う．仕様書から，変更要求によって追加・変更された機能に関連が深く，ソースコード上に現れそうな単語を抜き出して検索に用いるキーワードとする．このとき，単語の形をソースコード中で使われているものに合わせておく．例えば，status や control といった単語がソースコード上では stat や ctrl のように省略されることも多い．単語が不足するようなら，片仮名表記された単語や，ソースコード上からも実際に識別子等に使われている単語からもキーワードを収集する．

次に，ソースコード上から単語を検索する．検索する再に，ソースコード上の識別子のうち複数語からなると思われるものは単語ごとに分解してからキーワードと比較する．識別子は，複数の単語がアンダーバーで接続されたものや，キャピタライズされたものがほとんどであるので，容易に分解できる．関数ごとに単語の出現回数をカウントし，出現回数が設定された値以上ならば，その関数を候補として挙げる．

必要ならば，閾値と検索条件の調整の調整を行う．変更が必要な関数の候補の数を見ながら，閾値の調整，もしくは単語を追加をして，再度検索する．

## 4．適用

### 4.1 分析対象について

今回，分析を行ったソフトウェアは，組込みシステムのうち，ハードウェアを制御するディバイスドライバに相当する部分である．予測結果を評価するために，実際の製品の２つのバージョンのデータを使って分析した．

制御対象のディバイスは CPU のバスに接続され，メモリ空間上にマップされている．ディバイスへのアクセスは，メモリ上に割り当てられた FIFO やコントロールレジスタ，割り込みなどを介して行われる．ソフトウェアはハードウェアと共に開発され，機能追加のためにハードウェアの仕様も変更されている．

開発には C 言語が用いられ，規模は表１に示す通りである．

### 4.2 分析

あらかじめ，２つのバージョンを比較し，変更された部分のうち予測の対象とする機能に関連する箇所をリストアップしておく．実際の２つのバージョンの差異には，複数の変更が含まれているため，全てのソースファイルに対して diff を用いて差分を取り，手作業で対象とする機能に関する変更箇所のみを列挙した．表１に変更箇所を集計した結果を示す．集計に使ったデータは C のソースのみとし，ヘッダファイル等は除外している．なお数値は概数となっている．

| | 変更あり | 全体 |
|---|---|---|
| ファイル数 | 10 | 20 |
| 関数 | 20 | 300 |
| 行数(LOC) | 1000 | 30〜40k |

表１ 比較結果

変更されている関数の数は全体の１割以下であるが，多くのファイルに変更のあった関数が分布していることがわかる．変更の規模は行数だけ見た場合大きくないが，多くの箇所に分散しているため，修正する箇所を探す手間は無視できない．

次に，仕様書から，追加された機能に関する記述を探し，その周辺からソースコード上に現れそうなキーワードを抽出する．具体的には，追加された機能に関する単語の他に，追加された機能と同等の位置づけにある既存機能に関する単語などを用いることにした．例えば，mp3 形式のデータのみに対応しているソフトウェアに対して wma 形式のサポートを追加したい場合，mp3 に関連するキーワードを用いる．今回は合計で８単語を抽出した．

予測した結果から実際にソフトウェアを変更

した場合に，変更漏れがあると問題であるので，全ての関数が予測結果に含まれていることが望ましい．そのためには，精度が多少低下しても，再現率が１である必要がある．そこで，検索に使うキーワードの数と，再現率と精度の関係を調べ，どの程度のキーワード数で再現率が１になるかを調査した．

再現率(recall)と精度(precision)は以下の式により求められる．

$$再現率 = \frac{正しく検出した関数の数}{実際に変更された関数の数}$$

$$精度 = \frac{正しく検出した関数の数}{見つかった関数の個の数}$$

## 4.3 分析結果

抽出した８個のキーワードを用いて，いくつかの条件の上で，検索に使うキーワード数に対して再現率と精度がどのように変化するか調査した．

まず，出現頻度を考慮せずに，キーワードが１回でも出現する関数を検索し，検索に使うキーワード数と再現率と精度の関係を調査した．キーワードの数と再現率と精度の関係を図２に示す．

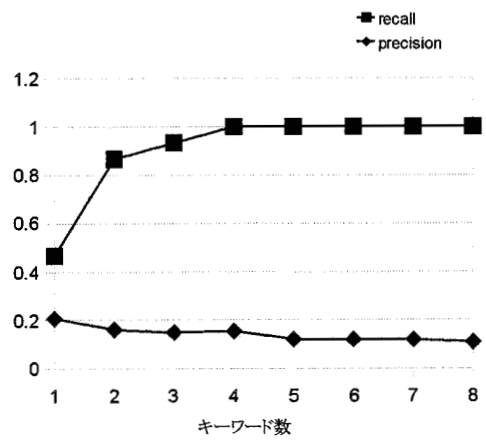


図２　キーワードが１回以上現れる関数

検索に使う単語数を増やせば，実際の変更箇所に該当する関数の多くが見つかるようになる．キーワード数を５個にした時点で全ての関数を見つけることが出来ている．さらにキーワードを増やすと，余分に候補に上がる関数が増えるため，精度が下がっていく．

キーワードが１回でも現れる関数を候補としてしまうと，多くの関数が候補に挙がってしまうため，精度がかなり悪い．キーワードの出現回数に閾値を設け，閾値以上になった関数のみを候補とすれば，制度が改善されるはずである．

次に，キーワードが５回以上含まれることを条件に検索した結果を示す．キーワードの出現回数は，単純にキーワードに含まれる単語が現れた回数を表し，同じキーワードが複数回現れてもカウントしている．

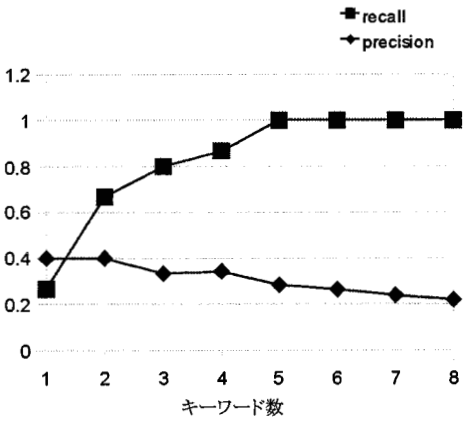


図３　キーワードが５回以上現れる関数

キーワードの出現回数を５回以上とした場合キーワードを増やした時の再現率の増加は緩やかになったが，再現率が１となるキーワード数は同じく５個であった．出現回数を考慮しなかった場合には，再現率が１になった時点での精度が0.12程度であったが，0.28程度まで改善した．このとき，全関数のうちの約20%の関数が候補として挙げられていた．

さらに閾値の出現回数を６回に増やした場合を図４に示す．

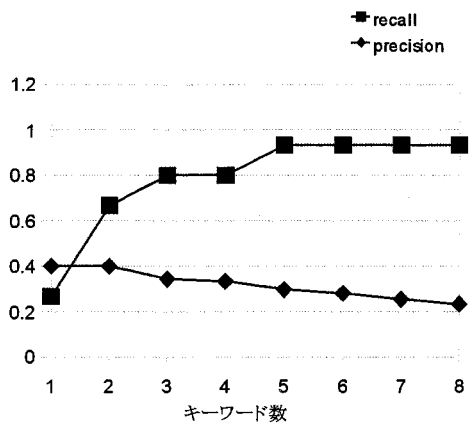


図４キーワードが６回以上現れる関数

再現率の増加はさらに緩やかになり，キーワードの出現回数の閾値を８個まで増やしても１にならなくなったが，精度の改善はあまり見られなかった．検索に使うキーワードをさらに多くすれば１になると考えられるが，精度は８個の時点より改善することはありえない．

表２に出現回数の閾値，再現率を１にするために必要なキーワード数，そのときの再現率と精度から求めたＦ値の関係を示す．

Ｆ値は以下の式より求められ，高いほど検索の性能が良いことを表す．

$$F\text{値} = \frac{2 \cdot (\text{精度} \cdot \text{再現率})}{\text{精度} + \text{再現率}}$$

| 出現回数(閾値) | 必要なキーワード | F値 |
|---|---|---|
| 1 | 4 | 0.27 |
| 2 | 5 | 0.24 |
| 3 | 5 | 0.32 |
| 4 | 5 | 0.38 |
| 5 | 5 | 0.44 |
| 6 | (8) | 0.37 |

表２

表から分かるように，閾値を変化させても，再現率を１にするために必要なキーワード数はほとんど変化していない．多くのキーワードは相互に関連が深く，単独で現れることはあまり無いためであると考えられる．

次に，キーワードの出現箇所を考慮して検索を行った．ある機能が複数の箇所に分散して実装されている場合，その機能に関するキーワードが条件文に現れることが多いと考えられるため，条件文等を考慮して検索を行えば，精度の向上が望めるはずである．以下に，機能「foo」に関する単純な条件文の例を示す．

```
if (foo==TRUE  && type == FUNC_FOO) {
  ..... // 機能 foo に関する処理
} else if (bar==TRUE && type == FUNC_BAR ) {
  .....
} else if .. {
  .....
}
```

条件文にはif文,switch-case文の他に，for文やwhile文の条件式も含んでいる．図５に検索対象を条件文に含まれるキーワードに限定した場合の精度と再現率のグラフを示す．

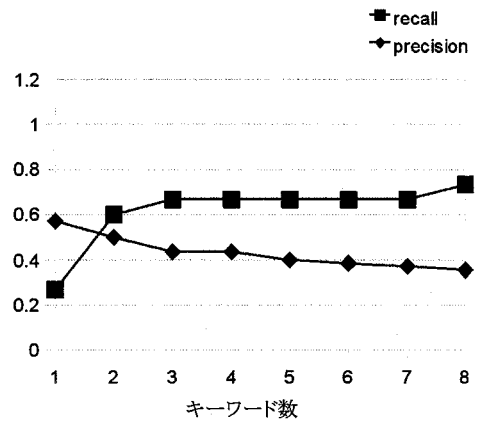


図５　条件文に含まれるキーワードで検索

キーワードの出現箇所を考慮した場合，考慮しない場合よりも精度は向上するが，キーワードを８個まで増やしても全ての関数を見つけることはできず，再現率が１にならなかった．このような結果になったのは，そもそも条件分岐を含まない関数が少なからず存在することや，分岐の条件として使われないキーワードも多いことが原因と考えられる．

### 4.4 考察

今回の分析の場合は，キーワード数が５個のときに最も良い検索結果となった．実際にこれから行う変更に対して適用する場合には，一番良いキーワード数を知ることは出来ないが，この結果から最適なキーワード数にばらつきは少ないと考えられるので，分析対象が変わっても近い値になる思われる．

条件文を使ってキーワードの出現箇所を考慮した場合，再現率が下がるが精度は向上したので，キーワードの出現箇所によって適切に重み付けすることで，再現率が１になった時点での精度を上げることが可能であると考えられる．

## 5．まとめと課題

### 5.1 まとめ

本稿では，組込みソフトウェアの変更開発において，仕様書とソースコードのトレーサビリティをサポートする手法を提案し，実データに基づいて評価した．対象ソフトウェアに関する知識がある程度あれば，仕様書等から得たキーワードによってソースコードの変更箇所をある程度絞れることを示した．対象ソフトウェアに関する知識が全く無い状態では，キーワードの選定が難しく，変更の規模が全くわからない場合は関数の数を見ながら検索のパラメータの調節するようなことはできないが，旧バージョンの開発者がそのまま開発を続ける場合には問題にならないであろう．

また，キーワード数や検索条件と予測の再現率や精度の関係についてのデータが得られた．条件にもよるが，再現率を１にするためのキーワード数はそれほど多くはならず，また精度の低下は緩やかであるので，必要十分なおおよそのキーワード数は予測可能であると考えられる．検索対象を条件文等に現れるキーワードに限定するなど，プログラムの構造を考慮して検索することで，予測の精度が向上することが確認できた．

### 5.2 課題

検索対象を条件文等に現れるキーワードに限定するなど，プログラムの構造を考慮して検索することで，予測の精度を上げることが確認できたが，再現率を上げることが難しくなるという問題がある．これはキーワードが出現する文脈によって，重み付けをすることで改善できと考えられるが，手法の有効な適用範囲がかなり狭くなってしまう可能性があるので考慮する必要がある．

本研究では，キーワードの抽出は開発者が手作業で行うことにしたが，ソースコードをある程度読まなくては，どのキーワードが使えるか判別がむずかしい．キーワードの抽出を自動化できれば，開発者への負担はより小さくなるはずである．単語と文書で確率ネットワークを構成し，半自動的に異なるドキュメント間の関連付けをする試みがJaneらの研究でなされている[11,12]．本研究で扱った仕様書とソースコードの間では，共通して用いられる単語の数があまり多くないため，精度の良い確率ネットワークが作れない可能性もある．ソースコード上の識別子等に使われそうな単語に関する同義語辞書等を用いることによって改善できる可能性はある．

近年，組込みソフトウェアの分野でもユースケースやシーケンス図等にUMLが導入されるケースも増えてきていると考えられる．UML等で表記された設計図も含めて変更予測を行うことも視野にいれる必要がある．また，組込みソフトウェアの種類や変更の内容は多様で，１つの手法でカバーできる範囲は限られている．他の開発事例に適用した場合には結果が大きく異なることも考えられるので，その調査も今後の課題である．

## 謝辞

本研究は株式会社半導体理工学研究センター(STARC)の支援のもとで行われた．

## 参考文献


[1] 高田 広章：「組込みシステム開発技術の現状と展望」, 情報処理学会論文誌, Vol.42, No.4 (20010415) pp. 930-938

[2] 小池 誠, 並木 美太郎, 岩澤 京子：分散環境における再利用性の高いオブジェクト作成を支援する共同開発環境の研究, 情報処理学会研究報告. ソフトウェア工学研究会報告, Vol.99, No.28 (19990318) pp. 41-48

[3] Jorma Taramaa, Munish Khurana, Pasi Kuvaja, Jari Lehtonen, Markku Oivo and Veikko Seppanen., Product-Based Software Process Improvement for Embedded Systems, EUROMICRO 1998.

[4] 渡辺博之, 渡辺政彦, 堀松和人, 渡守武和記：組み込み UML eUML によるオブジェクト指向組み込みシステム開発, 株式会社翔泳社, 2002

[5] Jane Cleland-Huang. Toward improved traceability of non-functional requirements. Proc. of International Workshop on Traceability in Emerging Forms of Software Engineering, (TEFSE) 2005.

[6] 後藤英斗，大久保弘崇，粕谷英人，山本晋一郎：文脈に基づいたソースプログラムとドキュメント間の識別子対応付け手法, 情報処理学会研究報告.ソフトウェア工学研究会報告, Vol.2005, No.29 (20050317) pp. 41-48

[7] 神戸英利，永松博子，三井浩康，小泉寿男：組込みソフトウェアの再利用を支援するモジュール間相互関連表示法, 情報処理学会研究報告. ソフトウェア工学研究会報告, Vol.2007, No.52 (20070528) pp. 71-78

[8] 大場勝, 権藤克彦：アスペクト指向を用いたドキュメント整理法の提案, 日本ソフトウェア科学会 第 7 回 プログラミングおよび応用システムに関するワークショップ, 2004

[9] 大場勝：OS 入門用の教材を事例とした，アスペクト指向によるソースコードとドキュメントの関連づけ, 北陸先端科学技術大学院情報科学研究科 修士論文, 2004

[10] Sapid, http://www.agusa.nuie.nagoya-u.ac.jp/person/Sapid/

[11] Jane Cleland-Huang , Raffaella Settimi , Chuan Duan , Xuchang Zou. Utilizing Supporting Evidence to Improve Dynamic Requirements Traceability, Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on.

[12] Jane Huffman Hayesm, et al Improving Requirements Tracing via Information Retrieval, Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International.