Vol.2018-SE-199 No.13
2018/7/18

テストスクリプトの自動生成における テスト入力値作成支援技術の提案

倉林 利行[1,a)]　切貫 弘之[1,b)]　丹野 治門[1,c)]

概要：アプリケーションを短期間で繰り返しリリースするためには，ソフトウェア開発全体の工数の 25%以上を占めているテストを自動化することが重要である．テストを自動化するためにはテストケースをスクリプトで記述する必要があるが，テストスクリプトの作成に工数がかかることが問題となっている．テストを自動化するために必要な工数を削減するために，クローリングをすることでテスト対象から仕様情報を自動で復元してテストに用いる手法が存在する．テスト対象を用意するだけで使用することができるが，一方でログイン画面のような特定の入力値を必要とする場合はクローリングができない．そのため入力値を必要とする場合は，人手で入力値を用意して事前にクローラに与えなくてはならない．しかし入力値を用意するためには，入力フォームをすべて探し出した上で，各入力フォームに対応する入力値を指定するフォーマットをマシンが読み取れる形式で作成する必要があり，工数がかかることが課題となっている．本論文では，テスト対象から入力フォームを自動で検出し，各入力フォームに対応する入力値を指定するフォーマットを自動生成することで，入力値を用意する工数削減を狙う手法を考案した．

A Proposal of Input Value Creation Support Technique for Automatic Generation of Test Scripts

Toshiyuki Kurabayashi[1,a)]　Hiroyuki Kirinuki[1,b)]　Haruto Tanno[1,c)]

1. はじめに

近年，アプリケーションに対するテスターのニーズの変化やアプリケーションのプラットフォームとなるソフトウェア，ハードウェアの進化のスピードが速いため，これらに対しアプリケーションを短期間で対応させ，一定の品質を確保しつつリリースしていくことが強く求められている．アプリケーションをリリースするにあたって，リリース前に適切に動作しているかどうかを確認するためのテストを行う．このとき，追加や変更があった機能をテストをするだけでなく，変更を加えていない，既存の機能のテストも実施する必要がある．既存の機能に対するテストを，回帰テストと呼ぶ．開発にかかる工数全体のうち，25%以上を回帰テストが占めている [1]．既存の機能の再テストという非生産的な回帰テストに工数が割かれることは，テスターのニーズへ対応するための新機能の追加等の生産的な活動の妨げとなっており，回帰テストの工数削減が求められている．回帰テストの工数を削減するための方法として，テストの自動化が挙げられる．テスト自動化のためのツールとして，実際に画面を操作して動作を確認するテストを自動化する Selenium WebDriver [2] 等のテスト自動実行ツールが存在する．テスト自動実行ツールによって，テストにおいて実施したい画面操作等の手順をツールが読み取れる形式のテストスクリプトで記述することで，自動実行することができる．しかしテスト自動実行ツールの利用率はわずか 12%とあるように，導入が進んでいないのが現状である [1]．テスト自動実行ツールの導入が進んでいないことの理由として，テストを自動実行するために必要なテストスクリプトの作成に工数がかかることが挙げられる．テストスクリプトの作成では，テストの手順を逐一プログ

[1] NTT ソフトウェアイノベーションセンタ, 東京都港区港南 2–13–34 NSS2 ビル 6F
[a)] kurabayashi.toshiyuki@lab.ntt.co.jp
[b)] kirinuki.hiroyuki@lab.ntt.co.jp
[c)] tanno.haruto@lab.ntt.co.jp

ラムとして記述した後に動作確認や修正等も行う必要があり，同一のテストを人手で実行する場合より工数がかかる．しかし，仕様変更によってボタンや入力フォーム等の画面の操作対象に変更があり作成したスクリプトが使用不可能になった場合，工数の元が取れなくなってしまう [3] [4]．よって追加の工数をかけてまでテストスクリプトを作成するかどうかの判断は難しく，画面操作を伴うテストの自動化が進んでいない現状がある．

テストを自動化するために必要な工数を削減するために，テスト対象から仕様情報を自動で復元してテストに用いる手法 [5–9]（以下リバースベーステストと呼ぶ）が存在する．例えば Dallmeier らの研究 [6] では，テスト対象のリンクやボタンを自動で押下しクローリングしていくことで画面遷移を抽出し，その画面遷移を実現するテストケースをテストスクリプトとして出力している．入力として動作するテスト対象のソフトウェアを与えるだけで使うことができるため，ハイパーリンクやボタンのみで構成されたテスト対象に対しては工数をかけずに画面遷移を網羅するテストスクリプトを自動生成することができるという利点がある．しかしログイン画面のように特定の入力値を必要とする場合はクローリングができない．そのため入力値を必要とする場合は，人手で入力値を用意して事前にクローラに与えなくてはならない [6,7,9]．人手で入力値を用意して事前にクローラに与えることを以下入力値補助と呼ぶ．入力値補助をするためにはテスターが入力フォームをすべて探し出した上で，各入力フォームに対応する入力値を指定するフォーマットをマシンが読み取れる形式で作成する必要がある．そのため業務システムのような特定の入力値を与えないと到達できない画面遷移が多いシステムに対しては入力値補助に多くの工数を費やすことになってしまい，リバースベーステストを現場導入する上での障壁となっている．

本研究では，テスト対象から入力フォームを自動で検出し，各入力フォームに対応する入力値を指定するために必要な id,xpath 等の情報が記載されたフォーマットを自動生成することで，入力値を用意する工数を削減する手法を提案する．提案手法は既存の入力値補助で実施していた，入力フォームをすべて探し出す作業を自動化できるだけでなく，各入力フォームに対応する入力値を指定するフォーマットも自動生成できるため，テスターは自動生成されたフォーマットに入力値を与える作業だけ実施することになり工数削減が可能となる．本研究の貢献は以下の 2 点である．

(1) テスト対象から入力フォームを漏れなく自動で検出し，各入力フォームに対応する入力値を指定するために必要な id,xpath 等の情報が記載されたフォーマットを自動生成することで，入力値を用意する工数を削減する手法を提案した．

(2) 3kloc の規模の Web アプリを用いて従来手法に対する提案手法の工数削減効果を評価し，約 71%の工数削減効果を確認した．

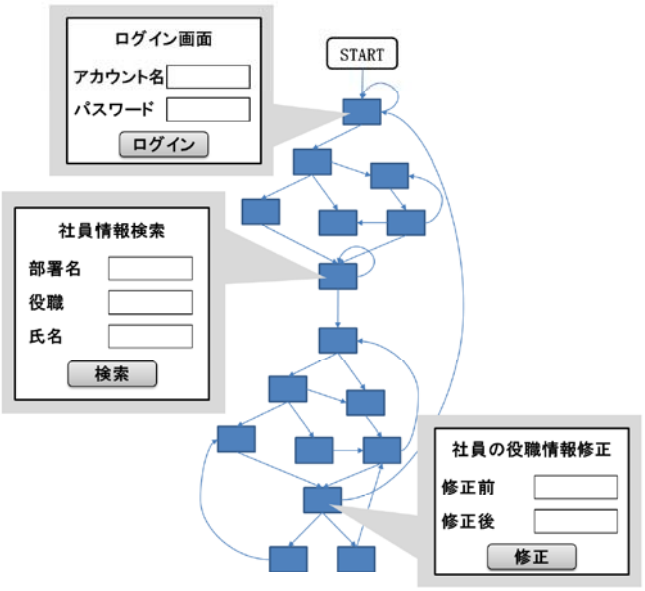

図 1　テスト対象の例

2. 従来の入力値補助方法と課題

従来の入力値補助を用いたリバースベーステストのフローを，図 1 の画面遷移で構成された社内システムを例として説明する．図 1 は入力値を必要する画面が 3 つある．1 つ目は「ログイン画面」，2 つ目は部署名，役職，氏名を入力して社員情報を検索する「社員情報検索画面」，3 つ目は変更前後の役職名を入力することで，登録された社員情報を修正する「社員の役職情報修正画面」である．上記 3 つの各画面で適切な入力値を入力しない限り，それぞれログイン成功後の画面，社員情報の検索に成功した画面，そして社員の役職情報の修正に成功した画面に遷移することができない．

2.1 従来の入力値補助方法のフロー

従来の入力値補助方法のフローを図 2 に示す．以下に各フローについて図 1 を用いて説明する．

(a) 入力値を指定するフォーマットの手動作成

入力：テスト対象
出力：入力値を指定するフォーマット
テスターは事前に入力フォームが存在する画面を調べ，各入力フォームに与える入力値を記入するフォーマットをクローラが読み取れる形式で作成する．図 1 において入力フォームが存在する画面を探すためには，実際にテスターがテスト対象を動かしながら 1 つ 1 つ画面を確認する．しかし入力フォームが存在する画面を漏れがないように探し

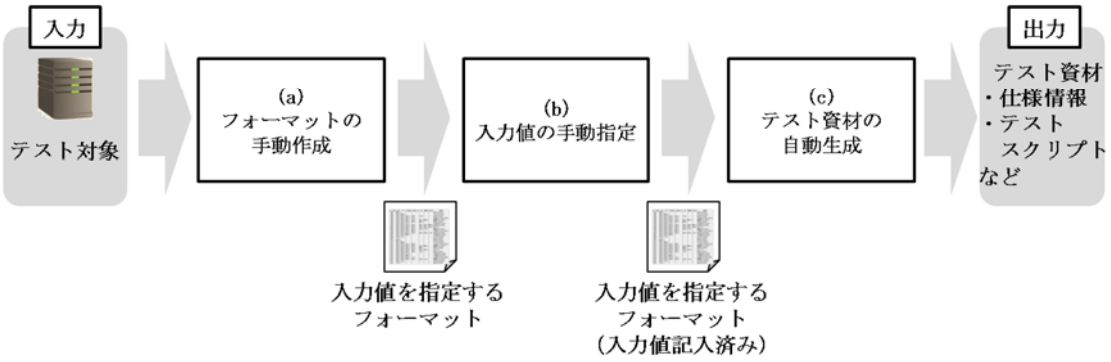

図 2　従来の入力値補助方法によるフロー

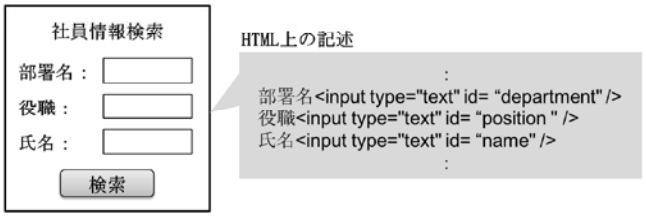

図 3　画面と HTML 表記の例

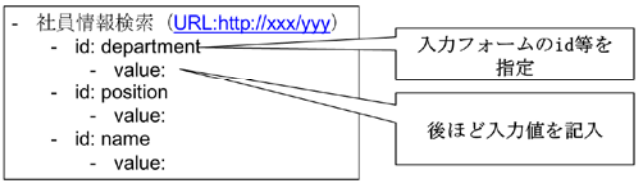

図 4　入力値を記入するフォーマットの具体例

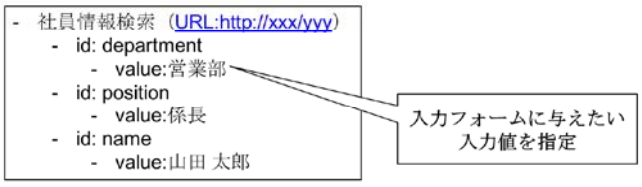

図 5　入力値を記入後のフォーマットの具体例

ていくためには，すべての画面にアクセスした上で，画面上に入力フォームがあるかどうかの確認を各画面に対して行う必要があり，工数がかかる作業となる．

続いてテスターは各入力フォームに与える入力値を記入するフォーマットを，クローラが読み取れる形式で作成する．クローラが読み取れる形式にするためには，入力フォームを一意に特定する情報（HTML の id 属性や xpath など）を用いる．例えば図 1 の社員情報検索画面の入力フォームの HTML 表記は図 3 のようになる．各入力フォームの HTML 上の表記を確認し，id の値を取得する．部署名を与える入力フォームの id は”department” になる．id が存在しない場合は xpath を用いることで入力フォームの情報を一意に特定することができる．続いて各入力フォームに与える入力値を記入するフォーマットを作成する．例えば図 1 の社員情報検索画面の入力値補助に必要なフォーマットは図 4 のようになり，id 等の入力フォームを一意に特定する情報と，その入力フォームに入力したい値を記載する箇所を用意する．**入力フォーム 1 つ 1 つに対して HTML を確認し，id 属性等の入力フォームを一意に特定する情報を漏れがないように収集しなくてはならないため，フォーマットの作成は工数がかかる作業となる．**

(b) 入力値の手動指定

入力：入力値を指定するフォーマット
出力：入力値を指定するフォーマット（入力値記入済み）
フォーマットを作成したあとは，テスターが画面遷移を網羅するために必要な入力値を記入していく．1 つの入力フォームに対して与えたい入力値が複数存在するときは，複数の入力値を記入する．図 4 のフォーマットに入力値を記入した具体例は，図 5 のようになる．

(c) テスト資材の自動生成

入力：入力値を指定するフォーマット（入力値記入済み），
　　テスト対象
出力：テスト資材（テストスクリプト等）
リバースベーステスト技術によって，テスト対象からテスト資材を自動生成する．クローリング中に入力フォームがあった場合，入力値を指定するフォーマットから該当する入力フォームを探し，記入された入力値を取得して入力フォームに与える．例えばクローリング中に社員情報検索画面に到達した場合，図 5 のフォーマットを調べて各 id に対応する入力フォームに value で指定されている値を入力する．テストスクリプトを生成する場合，画面遷移するために適切な入力値が与えられていれば画面遷移を網羅するようにクローリングすることができるため，テスト対象の画面遷移を網羅するテストスクリプトを生成することが可能である．例えば図 1 の社員情報検索画面において，検索を成功させるためのテストスクリプトは図 6 のようになる．図 6 は SeleniumIDE [10] の記法によるテストスクリプトの例である．入力値をいれる箇所に，図 5 で指定した入力値が記入されているのがわかる．このように入力値を指定するフォーマットによって，遷移させるために特定の入力値が必要な画面遷移においても，テストスクリプトを

```
        :

<!-- idがdepartmentの入力フォームに「営業部」と入力 -->
<tr>
        <td>type</td>
        <td>id=department</td>
        <td>営業部</td>
</tr>

<!-- idがpositionの入力フォームに「係長」と入力 -->
<tr>
        <td>type</td>
        <td>id=position</td>
        <td>係長</td>
</tr>

<!-- idがnameの入力フォームに「山田 太郎」と入力 -->
<tr>
        <td>type</td>
        <td>id=name</td>
        <td>山田 太郎</td>
</tr>

<!-- idがsearchのボタンをクリック -->
<tr>
        <td>clickAndWait</td>
        <td>id=search</td>
        <td></td>
</tr>

        :
```

図 6 テストスクリプトの例

生成することができる．

2.2 課題

2.1 節より，従来の入力値補助方法には以下の 2 つの課題があることがわかる．

課題 1　画面上に入力フォームがあるかどうかの確認を漏れがないようにすべての画面に対して行う必要がある

課題 2　入力値をクローラに与えるためにはマシンが読み取れるフォーマットで記述する必要があり，入力フォーム 1 つ 1 つに対して HTML を確認し，id 属性等の入力フォームを一意に特定する情報を収集しなくてはならない

例えば 100 画面，50 個の入力フォームから構成されるシステムにおいて，1 つの画面を探すために 3 分，画面に入力フォームがあるかどうか確認するために 1 分，入力フォーム 1 つに対してフォーマットを作成するために 1.5 分，入力値を与えるために 0.5 分かかるとすると，すべての入力フォームに 1 つの入力値を与えるためには，10 人時かかってしまう．リバースベーステストはテスト資材作成の工数削減を狙うための技術であるにもかかわらず，事前準備にこれだけの工数がかかってしまう可能性があることは，重要な課題である．

3. 提案する入力値補助方法

3.1 概要

提案手法では，事前にテスト対象をクロールすることで入力フォームの情報を収集し，各入力フォームに対して入力値を指定するためのフォーマットを自動生成することで，2.2 節であげた 2 つの課題を解決し，入力値補助にかかる工数を削減する．提案手法には以下の 2 つの特徴があり，それぞれが 2.2 節であげた 2 つの課題を解決していることがわかる．

特徴 1　入力フォームの情報をテスト対象から漏れなく自動で収集できるため，テスターが入力フォームを探す手間を省くことができ，課題 1 を解決できる

特徴 2　各入力フォームに対して入力値を指定するためのフォーマットを自動生成できるため，テスターはクローラに与えたい入力値を記入するだけで良いため，課題 2 を解決できる

3.2 提案する入力値補助方法のフロー

リバースベーステストにおいて提案する入力値補助を実施するフローは図 7 のようになる [6]．提案手法では，クローリングをすることでテスト対象から自動で入力フォームの情報を収集し，各入力フォームに対して入力値を指定するために必要な id,xpath 等の情報が記載されたフォーマットを自動生成する．しかし，画面遷移するために特定の値を必要とする入力フォームが存在する場合，その画面から先の画面に存在する入力フォームの情報を収集することはできない．そこで提案手法が出力した入力値を指定するためのフォーマットに対して画面遷移するための入力値を記入し，その入力値を元に再度クローリングを実施することで入力フォームを取得できる範囲を広げることができる．新しい入力フォームの情報が取得されなくなるまで上記を繰り返すことで，すべての入力フォームの情報が記載されたフォーマットを最終的に生成することができる．テスターは都度生成されるフォーマットに新しい入力フォームの情報が記載されていれば入力値を記入する作業のみを実施する．したがって 2.2 節であげた例と同様の例で考えると，提案手法では入力値を与えるための工数だけで済むため，0.4 人時ですべての入力フォームに 1 つの入力値を与えることができる．以下，同様に図 1 の例を用いて各フローについて説明する．

(a)' フォーマットの自動生成

入力：テスト対象，入力値を指定するフォーマット（入力値記入済み）

出力：入力値を指定するフォーマット

フォーマットの自動生成のフローを図 8 に示す．テスト対象をクローリングすることで入力フォームの情報を自動

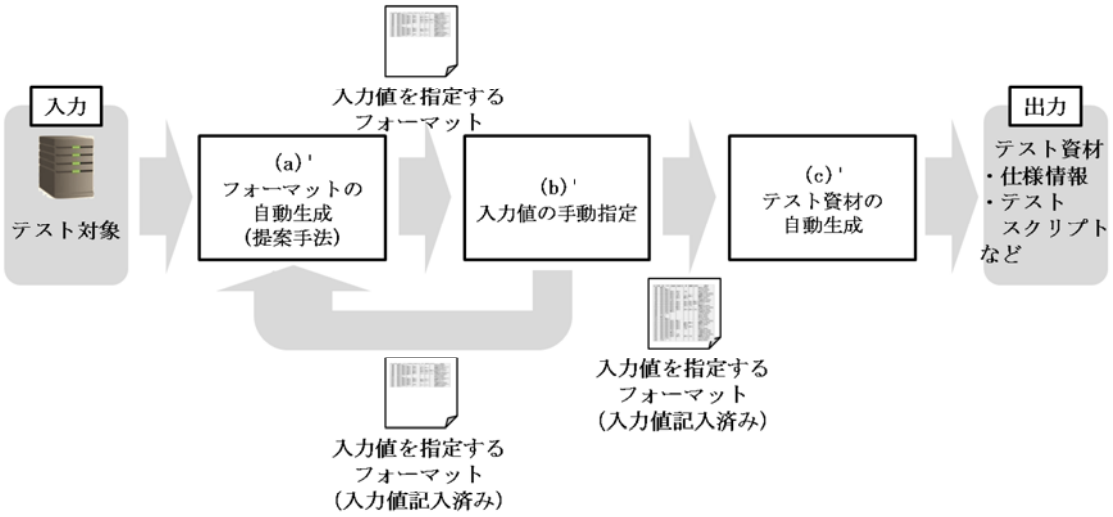

図 7　提案する入力値補助方法によるフロー

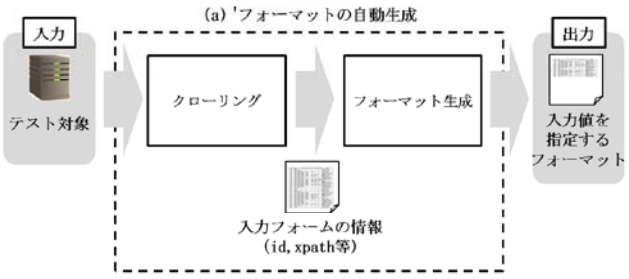

図 8　提案手法におけるフォーマットの自動生成のフロー

で収集し，入力値を補助するためのフォーマットを自動生成する．クローリング中に入力フォームがあった場合，入力値を指定するフォーマットから該当する入力フォームを探し，記入された入力値を取得して入力フォームに与える．フォーマットには id 等の入力フォームを一意に特定する情報だけでなく，テスターが見たときにどの入力フォームかわかりやすくするために，ラベルの情報も記載する．図 1 の例においては，初回のクローリング中はログイン画面から先に遷移することができないため，図 9 のフォーマット 1 を出力することになる．「(b)' 入力値の手動指定」でフォーマット 1 に入力値が与えられた 2 回目のクローリングではログイン画面から先に遷移できるようになっているが，次は社員情報検索画面から先に遷移できず図 9 のフォーマット 2 を出力する．フォーマット 2 に入力値が与えられた 3 回目のクローリングでは図 9 のフォーマット 3 を出力し，すべての入力フォームが存在する画面に対する入力値を補助するためのフォーマットが完成することになる．

(b)' 入力値の手動指定

2.1 節の (b) 入力値の手動指定と同様にフォーマットに入力値を記入する．記入が完了したら，(a)' フォーマットの自動作成に入力として渡してクローリングを再実行する．入力値が未記入の箇所が存在しない場合は，新しく発見された入力フォームが存在しないということであり，そのフォーマットを完成版として (c)' テスト資材の自動生成に入力として与える．

(c)' テスト資材の自動生成

2.1 節の (c) テスト資材の自動生成と同様である．

4. 評価

提案手法の評価方法及び評価結果を示し，考察を述べる．

4.1 方法

提案手法では，事前にテスト対象をクロールすることで入力フォームの情報を収集し，各入力フォームに対して入力値を指定するためのフォーマットを自動生成することで，入力値補助にかかる工数を削減することが狙いである．したがって本評価では工数を評価観点とし，入力値が記入された状態の入力値を指定するフォーマットを作成するまでにかかった工数を従来手法と提案手法で比較する．つまり従来手法では図 2 における (a),(b) にかかった工数，提案手法では図 7 における (a)',(b)' にかかった工数の比較となる．テスト対象で用いたシステムは 3kloc の規模の Web アプリであり，全 4 画面のうち，2 画面に入力フォームが必要な画面が存在する．1 つ目の画面はログイン画面で ID とパスワードを必要とし，2 つ目の画面は名前やメールアドレスを入れる入力フォームなど，全部で 11 個数の入力フォームが存在する登録画面である．

4.2 結果

評価結果を 1 に示す．提案手法では従来手法に比べて約 71%の工数削減に成功した．提案手法ではクローリングを 3 回実施し，1 回目はログイン画面の入力フォームのフォーマット，2 回目は登録画面の入力フォームのフォーマットを出力した．3 回目のクローリングではフォーマッ

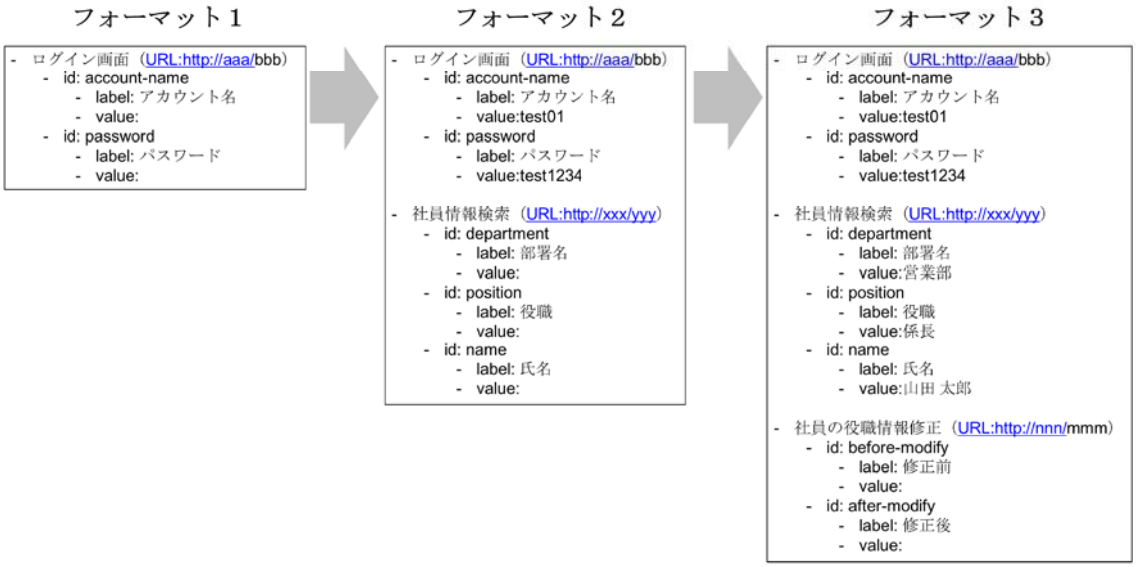

図 9 提案手法が自動生成するフォーマットの進化

表 1 従来手法と提案手法の工数比較

	フォーマット作成の工数 [人秒]	入力値の指定の工数 [人秒]	合計の工数 [人秒]
従来手法	196	69	265
提案手法	0	76	76

トに更新が見られなかったため，終了した．また提案手法のクローリングにかかった時間は 1，2，3 回目それぞれ 11 秒,43 秒,43 秒の合計 97 秒だった．

4.3 考察

提案手法が従来手法と比較して約 71%の工数削減に成功した理由として，提案手法ではフォーマットを自動生成おり，フォーマット作成に工数がかからなかったためだと考えられる．また入力値を指定するための工数は従来手法と提案手法でほぼ同一だった．提案手法でも従来手法でも，フォーマットに入力値を記入していくという作業に違いはなく，工数に差が出なかったと考えられる．以上の結果より入力フォームの数が多くフォーマットの規模も大きくなるようなシステムにおいて，提案手法による工数削減効果がより向上すると予想できる．一方で，提案手法では 3 回のクローリングを実施しており，マシンタイムが 97 秒かかった．テスト対象の規模が大きくなるにしたがって，クローリングの回数とマシンタイムも増加すると考えられる．提案手法は短期繰り返し開発での使用を想定しており，一度アプリケーションをリリースしてから次のリリース前のテストまでにテスト資材を生成できれば問題ないと考えている．業務システムのような巨大なシステムにおいて本提案手法を実行したときにかかる時間が，上記の次のリリースまでの時間の範囲に収まるかどうかについては今後評価していきたい．

5. まとめ

本研究では，テスト対象から入力フォームを漏れなく自動で検出し，各入力フォームに対応する入力値を指定するために必要な id,xpath 等の情報が記載されたフォーマットを自動生成することで，入力値を用意する工数を削減する手法を提案した．提案手法は既存の入力値補助で実施していた，入力フォームをすべて探し出す作業を自動化できるだけでなく，各入力フォームに対応する入力値を指定するフォーマットも自動生成できるため，テスターは自動生成されたフォーマットに入力値を与える作業だけ実施することになり工数削減が可能となる．3kloc の規模の Web アプリを用いた評価によって，提案手法では従来手法に比べて約 71%の工数削減できることがわかった．今後は業務システムを意識したより大規模なシステムに対して実験を実施することで，提案手法の実用性の評価を行いたい．

参考文献

[1] : ソフトウェアメトリックス調査 2012　報告書, http://www.juas.or.jp/servey/library/pdf/12swm.pdf.
[2] : Selenium WebDriver, http://www.seleniumhq.org/projects/webdriver/.
[3] Fewster, M.: システムテスト自動化 標準ガイド，翔泳社 (2014).
[4] Leotta, M., Clerissi, D., Ricca, F. and Tonella, P.: Capture-replay vs. programmable web testing: An empirical assessment during test case evolution, *2013 20th Working Conference on Reverse Engineering (WCRE)*, pp. 272–281 (online), DOI:

10.1109/WCRE.2013.6671302 (2013).
[5] Duda, C., F. G. K. D. M. R. and Zhou, C.: AJAX Crawl: Making AJAX Applications Searchable, *In IEEE 25th International Conference on Data Engineering*, pp. 78–89 (2009).
[6] Dallmeier, V., Pohl, B., Burger, M., Mirold, M. and Zeller, A.: WebMate: Web Application Test Generation in the Real World, *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, pp. 413–418 (online), DOI: 10.1109/ICSTW.2014.65 (2014).
[7] Schur, M., Roth, A. and Zeller, A.: Mining behavior models from enterprise web applications, *ESEC/SIGSOFT FSE* (2013).
[8] Choudhary, S. R., Prasad, M. R. and Orso, A.: CrossCheck: Combining Crawling and Differencing to Better Detect Cross-browser Incompatibilities in Web Applications, *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pp. 171–180 (2012).
[9] Mesbah, A., van Deursen, A. and Lenselink, S.: Crawling Ajax-Based Web Applications Through Dynamic Analysis of User Interface State Changes, *ACM Trans. Web*, Vol. 6, No. 1, pp. 3:1–3:30 (online), DOI: 10.1145/2109205.2109208 (2012).
[10] : Selenium IDE, http://www.seleniumhq.org/projects/ide/.