Automated Testing Featuring Prototype Generation from Harvested Requirements Specification

Nawin Phuangphoo and Yachai Limpiyakorn

Department of Computer Engineering, Chulalongkorn University, Bangkok, Thailand
Nawin.P@student.chula.ac.th, Yachai.L@chula.ac.th

Abstract. Prototyping is a common technique suggested for requirements validation during the early phase of software project. However, prototype construction is resource consuming. Moreover, the evolving prototype due to requirements change may cause inconsistency among associated artifacts. This paper thus presents an approach to automating the construction of prototypes from harvested requirements specification. The implemented component is part of Requirements Harvester— RH, which is the system to facilitate integrating quality control activities with requirements management ability provided by a traceability matrix. The generated prototype is self-test and capable of maintaining the consistency of related work products. The presented automation approach promotes the working smarter environments that could improve process capability and performance. The findings from the comparisons of the manual and the proposed methods reported the outperformance of the latter.

Keywords: Prototyping, Content harvester, User Interface Testing, and Software Process Improvement.

1 Introduction

Requirements are raw materials for developing software. Throughout the life cycle processes, requirements are transformed into design blueprints and code, respectively. Therefore, the early determination of what the user really wants can result in faster and less expensive software, because changes cost exponentially more to implement if they are detected later in development [1].

For mature software processes, prior to formal hand-off from the phase of requirements analysis, prototyping is one of the methods or techniques typically used to demonstrate the essential features of the system being developed so that the common understanding among the developers, the end users, and other relevant stakeholders could be established. The valid requirements are the basis of the final quality products, as well as faster and less expensive software projects.

Prototyping can be considered as a necessary auxiliary step that can help increase speed, improve quality and cut cost. Moreover, it can be used to test user interface flows or as a means of reducing the risk caused by wrong user interfaces, which are usually a most visible or most risky part. Since users know the problem domain better than anyone in the development team, increased user involvement can result in the final product that is more likely to satisfy the users' desire for look, feel and performance [1].

Today, there are many tools to facilitate creating a prototype. For example, screen generators that provide the drag and drop feature to help create prototypes easier. However, the prototype created simply shows users the system that does not function. It merely shows what the screens may look like. Screencast (or video screen capture) is a digital recording of computer screens which may contain audio narration. However, there are no prototyping tools that facilitate creating a prototype directly from the requirements specification, support self-test, and maintain consistency when requirements change. This research thus presents a method to harvest the contents of requirements specification, and to generate the prototype with the features of self-testing and self-adapting to requirements changes.

2 Content Harvester

In literature, the purpose of Content Harvester [2] is to unleash the information in a collection of unstructured, formatted documents that follow a similar pattern, and to make that information available for publishing in any open format. Typically, the process of content harvester starts with: 1) converting the source document into XML format; 2) specifying the regions of content that are of interest in terms of textual markers or tagging based on user-defined names; 3) locating the target tags; and 4) extracting the associated contents. The resulting output is XML contents which can be queried for information exchanges among applications or platforms.

A specification contains some concrete information to link between requirements and programs. User interfaces are one of the information contained in a specification. Based on the notion of content harvester, software requirements can be extracted from the location detected by XML tags. The extracted data will then be processed for the prototype construction.

3 Design and Implementation

Our research group has implemented a system called Requirements Harvester (RH), of which the purpose is to facilitate the integration of quality control activities to requirements management ability provided by a traceability matrix [3], which is used for tracking any change and reflecting it properly from the initiation point to implementation. It can also detect the indirect or transitive relationships as well as indicate the suspect requirements that might be affected if a linked requirement is changed or deleted. Moreover, the input into the traceability matrix [3] is automated that could reduce human errors and resource consumption.

The architecture of RH is illustrated in Fig. 1. The current major components consist of requirements traceability matrix [3], the input preprocessing component [4], the user acceptance tests generator [5], and the prototype generation component which is the focus of this research as the enhancement of RH.

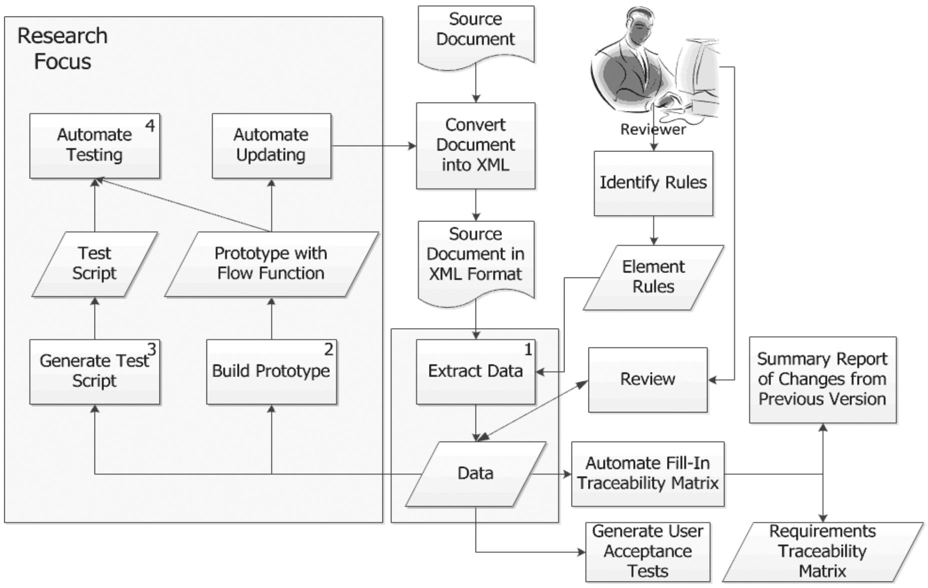

Research Focus
Automate Testing 4
Automate Updating
Test Script
Prototype with Flow Function
Generate Test Script 3
Build Prototype 2
Source Document
Convert Document into XML
Source Document in XML Format
Extract Data 1
Data
Reviewer
Identify Rules
Element Rules
Review
Summary Report of Changes from Previous Version
Automate Fill-In Traceability Matrix
Generate User Acceptance Tests
Requirements Traceability Matrix

Fig. 1. Architecture of prototype generation component of Requirements Harvester

The prototype generation component is developed as Java application. It comprises four major modules as described in the following subsections.

3.1 Extract Useful Information

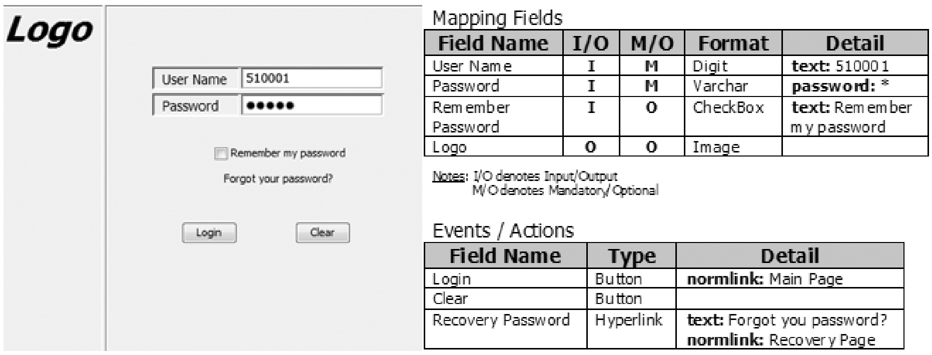
It is required that the input requirements specification contains a section of screen design and screen element descriptions, examples as shown in Fig. 2. The screen design can be handwritten, printed, image capture, etc. Screen images are used as the screen templates for generating user interfaces, while each screen element description provides details including list of elements (e.g. button, checkbox, drop-down list, link etc.) associated with properties, and transitions to other screens. The contents harvested from requirements specification are stored in DOM (Document Object Model) tree which enables direct data access and allows data modification. Moreover, it enables automatic updates onto requirements specification when there are minor changes to the associated prototype.

3.2 Build Prototype

A prototype contains two main components: 1) a set of user interfaces, and 2) the control flow. Each user interface (UI) is generated based on the screen image template which guides the developer to locate the screen elements. The step of generating UI requires human intervention. The type and properties of each screen element are known from the SRS analysis and extraction in the prior step. The program will match each element and its properties to the user-specified area. Some input types and exceptions are pre-defined for text fields. The program can examine these criteria with the event flow type and responds to it correctly. Additional exception handler may be triggered as needed in case of exception flow transition. At this stage, we obtain a set of user interfaces that do not function, but show what the screens may look like.

The prototype generator supports conventional types of UI element and argument. Example types of argument are shown in Table 1. The argument detail can be automatically filled in by the information extracted from requirements specification or manually input by the user on the screen template. Typical UI elements supported include: TextField, TextArea, PasswordField, RadioButton, CheckBox, Button, DropDownList, List, ScrollPane, Label, Panel, Slider, Spinner, ProgressBar, DateChooser, FileChooser, ColorChooser, etc.

Next, the component will match the element, such as button and link, with the control flow or transition analyzed from the source document. As a result, a prototype that does function is obtained. Another feature of the prototype generation component is that it provides automatic updates on requirements specification once the current prototype has been modified due to minor changes. The procedure is presented in Fig. 3. This feature promotes the consistency among work products produced in a project.



User Name 510001
Password ●●●●●

Mapping Fields

Field Name	I/O	M/O	Format	Detail
User Name	I	M	Digit	**text:** 510001
Password	I	M	Varchar	**password:** *
Remember Password	I	O	CheckBox	**text:** Remember my password
Logo	O	O	Image	

Notes: I/O denotes Input/Output
M/O denotes Mandatory/Optional

Events / Actions

Field Name	Type	Detail
Login	Button	**normlink:** Main Page
Clear	Button	
Recovery Password	Hyperlink	**text:** Forgot you password? **normlink:** Recovery Page

Fig. 2. Example screen design and screen element description contained in input SRS

3.3 Generate Test Script

The self-test feature is added to the prototypes generated. That is, the user interfaces and their flows can be automatically generated from requirements specification. Walking through the user interfaces to validate a set of requirements could be automated by following the control flow. State machine testing is introduced here. It is assumed that a state is assigned to each page or an element, all of which are then explored. Example state diagram of the screen design in Fig. 2 is shown in Fig. 4.

Applying the algorithm of graph traversal will generate a set of test paths, example as shown in Fig. 4. Each test path represents a test case. A set of test cases comprise a test script. A test script contains trigger component, trigger method (action), source page and destination page. It is defined based on the deterministic finite state machine, so called Variable Finite State Machines or VFSM [6]. VFSM can be converted to other equally finite state machines and can be applied for user-interface testing. The main component of this state machine is small and natural compared to others. In addition, it is more flexible, much faster, and more accurate because of its design for representing program procedure.

Table 1. Example of supported UI argument types

Type	Detail
Position	User clicks to notify start position
Size of Element	User clicks upper left and lower right co-ordinate to frame the area size
Text of Element	Extract from SRS
Drop Down List Items	Extract from SRS
Regular Expression for Input Validation	Extract from Format column in Mapping Fields Table (Fig. 2)
Date Format in Date-Chooser	Extract from SRS or default format applied
Tool Tip Text	Extract from SRS or specified by human
Input/Output Type	Notified by I/O column in Mapping Fields Table (Fig. 2)
Required	Notified by M/O column in Mapping Fields Table (Fig. 2)
Normal Link	Extract from Detail column in Events/Actions Table (Fig. 2)
Exceptional Link	Extract from Detail column in Events/Actions Table (Fig. 2)
Label Text	Extract from SRS or specified by human
Label Position	Default position determined by the program
Comment	Extract from SRS or specified by human
Icon Image	Extract from SRS or specified by human

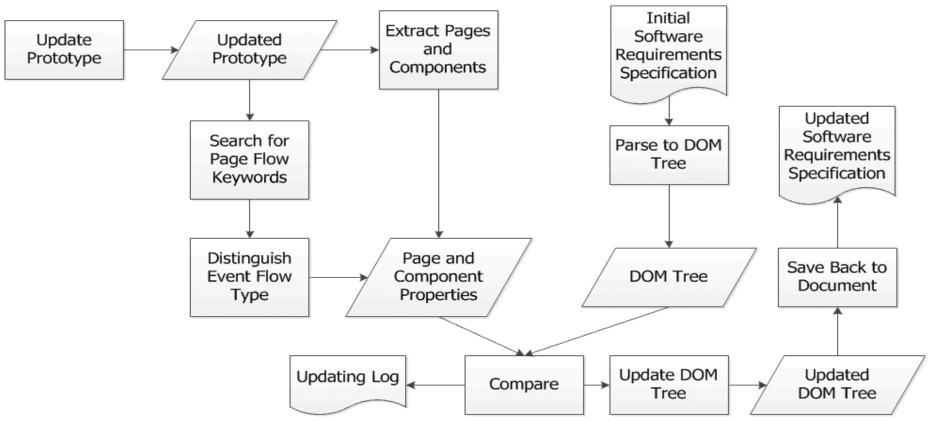
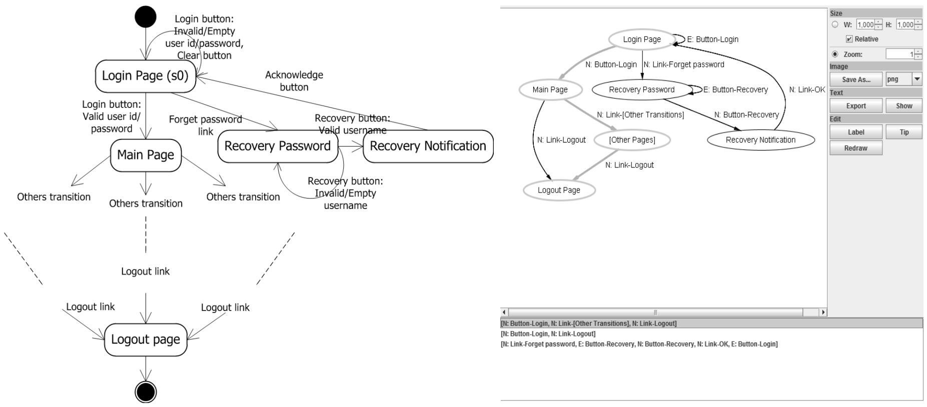

Update Prototype
Updated Prototype
Extract Pages and Components
Initial Software Requirements Specification
Search for Page Flow Keywords
Parse to DOM Tree
Updated Software Requirements Specification
Distinguish Event Flow Type
Page and Component Properties
DOM Tree
Save Back to Document
Updating Log
Compare
Update DOM Tree
Updated DOM Tree

Fig. 3. Steps to update SRS when prototype has been updated due to minor changes



```
[N: Button-Login, N: Link-[Other Transitions], N: Link-Logout]
[N: Button-Login, N: Link-Logout]
[N: Link-Forget password, E: Button-Recovery, N: Button-Recovery, N: Link-OK, E: Button-Login]
```

Fig. 4. Example state diagram and its associated graph and test paths

3.4 Automate Testing

This research opts to automate testing with the insertion of native system input events. The reviewer can easily inspect a particular procedure with automated walkthrough. Native system input events consist of Mouse move, Mouse press, Mouse release, and Key code send that emulate human behaviors. A set of operations are defined on individual sequence of events. Each operation can be regarded as human interaction.

3.5 Evaluation

Some experiments were conducted to demonstrate the significant difference of time usage between the manual against the proposed automation approach to prototyping. The results strongly support working smarter strategy or automation approach. Table 2 and 3 summarize the method type of each step comprising the procedure.

Table 2. Tasks of creating prototype and testing (initial)

Procedure	Manual	Automated
Documentation (SRS)	Manual	Manual
Review Document	Manual	Manual
Analyze Data	Manual	Automated
Import Document	-	Automated
Configure Input Specification	-	Manual (or Default)
Input Data into System	Manual	Automated
Define Element Flow	Manual	Manual
Create Prototype	Manual	Automated
Test Prototype	Manual	Automated (+Human observation allowed)

Table 3. Tasks of modifying prototype and retesting

Procedure	Manual	Automated
Review Document	Manual	Automated
Analyze Data	Manual	Automated
Import Document	-	Automated
Input Data into System	Manual	Automated
Define Element Flow	Manual	Automated
Modify Prototype	Manual	Automated
Test Prototype	Manual	Automated (+Human observation allowed)

Each task was performed and the usage of time was recorded. The time usage is compared in the dimensions of manual against automation, various sizes of requirements, and manual change against automated change, as shown in Table 4 (Note that Change2 denotes the second time of requirements change). The findings of Table 4 reported the much less time spent with automation approach for all scenarios. The percentage of differences is about 60%-70%.

Table 4. Comparisons of time usage (minute)

Number of requirements	25	50	100	150	300
a) Manual	397	794	1588	2381	4763
b) Automated	145	273	530	786	1555
c) Manual Change	550	1100	2200	3300	6600
d) Automated Change	174	328	638	947	1875
e) Manual Change2	703	1406	2813	4219	8438
f) Automated Change2	202	384	746	1109	2196
Diff Initial (%)	63	66	67	67	67
Diff Change (%)	68	70	71	71	72
Diff Change2 (%)	71	73	73	74	74

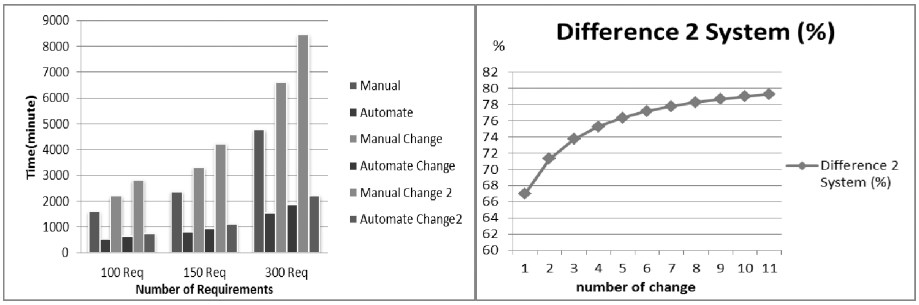


Fig. 5. a). Comparison of time usages between manual and automated prototyping b). Differences of time usages when more changes

The differences of time usages between manual and automation at the sizes of requirements: 100, 150, and 300, is presented with the bar chart in Fig. 5a). The tendency of the differences of time usages between manual and automated procedure after 10 changes at the size of 150 requirements is also illustrated in Fig. 5b). The result shows that automation is faster than manual around 65-80%. The greater number of change, the more difference of time usages was reported.

4 Conclusion and Future Work

User Interface Prototyping, or horizontal prototype, provides a broad view of an entire system or subsystem, focusing on user interactions rather than low-level system functionality [7]. A prototype is supposed to be developed quickly to explore the system requirements and user interface design early. Based on Content Harvester, the implemented prototype generator could reduce cost and effort by generating prototypes directly from the requirements specification, as well as help manage the consistency between the requirements specification and the prototype system due to requirements changes. This can be considered as the preventive approach to promoting conformance to requirements that will result in quality software product.

Experiments were conducted to demonstrate the efficiency of automation approach to prototyping compared to the manual method. The results showed significant reduction of time when constructing and modifying the prototype automatically.

References

1. Overmyer, S.P.: Revolutionary vs. Evolutionary Rapid Prototyping: Balancing Software Productivity and HCI Design Concerns. In: The Fourth International Conference on Human-Computer Interaction, pp. 303–307 (1991)
2. Srivastava, B., Chang, Y.: Business Insight from Collection of Unstructured Formatted Documents with IBM Content Harvester. In: ACM International Conference on Management of Data, pp. 73–78. ACM, New York (2010)
3. Soonsongtanee, S., Limpiyakorn, Y.: Enhancement of Requirements Traceability with State Diagrams. In: 2nd International Conference on Computer Engineering and Technology, vol. 2, pp. V2-248–V2-252 (2010)
4. Phopan, Y., Limpiyakorn, Y.: Approach to Automating Input Data for Requirements Traceability Matrix. In: Proceedings of the National Graduate Research Conference 2011, pp. 1033–1042 (2011)
5. Ieamsaard, C., Limpiyakorn, Y.: On Integrating User Acceptance Tests Generation to Requirements Management. In: International Conference on Information Communication and Management, vol. 16, pp. 248–252 (2011)
6. Shehady, R.K., Siewiorek, D.P.: A Method to Automate User Interface Testing Using Variable Finite State Machines. In: 27th International Symposium on Fault-Tolerant Computing, pp. 80–88. IEEE Press, Washington, DC (1997)
7. Nielsen, J.: Usability Engineering. Morgan Kaufmann, San Francisco (1993)