Spring 5-3-2016

Why Do Record/Replay Tests of Web Applications Break?

Mouna Hammoudi
University of Nebraska - Lincoln, mouna@cse.unl.edu

WHY DO RECORD/REPLAY TESTS OF WEB APPLICATIONS BREAK?

by

Mouna Hammoudi

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Gregg Rothermel

Lincoln, Nebraska

May, 2016

WHY DO RECORD/REPLAY TESTS OF WEB APPLICATIONS BREAK?

Mouna Hammoudi, M.S.

University of Nebraska, 2016

Advisor: Gregg Rothermel

Software engineers often use record/replay tools to enable the automated testing of web applications. Tests created in this manner can then be used to regression test new versions of the web applications as they evolve. Web application tests recorded by record/replay tools, however, can be quite brittle; they can easily break as applications change. For this reason, researchers have begun to seek approaches for automatically repairing record/replay tests. To date, however, there have been no comprehensive attempts to characterize the causes of breakages in record/replay tests for web applications. In this work, we present a taxonomy classifying the ways in which record/replay tests for web applications break, based on an analysis of 453 versions of popular web applications for which 1065 individual test breakages were recognized. The resulting taxonomy can help direct researchers in their attempts to repair such tests. It can also help practitioners by suggesting best practices when creating tests or modifying programs, and can help researchers with other tasks such as test robustness analysis and IDE design.

Acknowledgements

I would like to thank my advisor and my role model, Gregg Rothermel for making me discover the universe of research, dedicating to me all the time I needed and answering all of my questions and concerns. I would like to thank him for supervising me, guiding me throughout this journey and giving me this amazing opportunity. This work would not have been possible without him.

Also, I would like to thank Myra Cohen and Sebastian Elbaum for offering me their time, serving on my master's committee and for giving me feedback about this work.

I would like to thank my sister, my mum and my dad for encouraging me, motivating me and believing in me, this thesis would not have been possible without them.

Moreover, I would like to thank Wayne Motycka for installing some of the objects of study used in this thesis.

This work has been partially supported by the National Science Foundation through award IIS-1314365.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Web application developers frequently employ record/replay tools to enable the automated testing of their applications. A record/replay tool for web applications captures inputs and actions (mouse clicks, keyboard entries, navigation commands, etc.) that occur as a web application is utilized. During playback, these inputs and actions are re-delivered to the browser engine. The importance of such tools is underscored by the number that exist in the research and commercial realms, including CasperJS [1], CoScripter [2], Dolos [3], Jalangi [4], Mugshot [5], Sahi [6], Selenium [7], Sikuli [8], TestMaker Object Designer [9], UFT (formerly Quick Test Professional) [10], WaRR [11], and Watir [12].

An advantage of creating web application tests via record/replay tools involves the ability to reuse them to regression test the web applications as they evolve. However, web application tests created by record/replay tools can be quite brittle in the face of system evolution. Changes as simple as renaming a page element or altering the choices in a drop-down list can cause such tests to break and require repair.

For this reason, researchers [13, 14] have recently begun to create techniques for automatically repairing the input and action sequences (hereafter referred to simply as

"tests") created for web applications by record/replay tools. Other research [15, 16, 17] has focused on improving the robustness of such tests. The authors of these papers have acknowledged the brittleness of tests and have singled out specific test constructs that are particularly problematic. There have been no comprehensive attempts, however, to characterize—via observations of actual evolving web applications—the causes of *test breakages*, [1] much less to measure the frequencies at which those different causes occur. Understanding these issues is arguably a prerequisite for test repair, as well as for other important tasks such as creating maintainable tests and designing record/replay tools.

In this work, we take steps to provide this understanding [2]. We began by collecting 300 releases/commits of five open-source web applications and constructed test suites for the earliest releases/commits of these applications using a record/replay tool. We then iteratively applied the tests to subsequent releases/commits of each application, collecting data on the causes of the failures observed. Given test breakages on a given version V, we then repaired broken tests and augmented the test suite so that repaired tests and tests of new functionality could be carried forward to the version following V. This process yielded data on 722 individual test breakages.

Using the data we gathered, we developed a taxonomy of the causes of test breakages that categorizes all of the breakages observed on the applications we studied. We then gathered 153 releases/commits of three additional web applications and applied the foregoing process to them as well; this yielded data on 343 additional test breakages. We analyzed these in light of our taxonomy and were able to accommodate

[1] We define a *test breakage* as the event that occurs when a test that used to work on a web application ceases to work on a new version of that web application, due to a change in the web application

[2] Portions of this work have appeared previously in [18]

all of them without further changing the taxonomy; this provides evidence that our taxonomy may be more generally applicable.

Our taxonomy (which to our knowledge is the first taxonomy of web test breakages) is useful in several ways. First, it can assist researchers interested in test repair and help them prioritize their efforts in favor of more common breakage types. Second, it can help engineers avoid the need for test repairs by alerting them to changes that may break tests. Third, it can inform the design of automated techniques for detecting problematic constructs in tests and replacing these with less brittle constructs. Fourth, it can inform the design of better IDEs for creating web applications, such as IDEs that help engineers avoid or prevent test breakages, or automatically repair tests as application modifications are made. Finally, our taxonomy can inform the design of test record/replay tools themselves.

The remainder of this document is structured as follows. Chapter 2 presents background information. Chapter 3 describes the processes we used to collect data and construct the taxonomy. Chapter 4 presents our taxonomy and provides examples of causes of test breakages corresponding to its categories. Chapter 5 presents data on the frequency of breakage categories and on the applicability of the taxonomy to three additional web applications. Chapter 6 discusses ways in which our taxonomy and associated data can be useful. Chapter 7 describes related work and Chapter 8 concludes.

Chapter 2

Background

Record/replay tools allow test engineers to capture sequences of inputs and actions applied to a web application's GUI. The recording process creates a test script that can then be replayed on the application.

2.1 Record/Replay Tools

As an example, see Figure 2.1. The figure depicts the user interface for one form in a web application used to register a car on a college campus, and includes (from top to bottom) a text field, a checkbox, a pair of radio buttons, a dropdown list, a submit button and a link. The code for this web page consists of HTML and Javascript and is shown in Figure 2.2. Not visible in the GUI but present in the code is a Javascript alert that causes a popup box to appear if the user attempts to submit an incomplete form.

To further illustrate the record/replay process, Table 2.1 shows a test created by Selenium IDE via the application of a sequence of inputs and actions to the registration web page. Each input or action causes a *Selenese command* to be inserted into the test

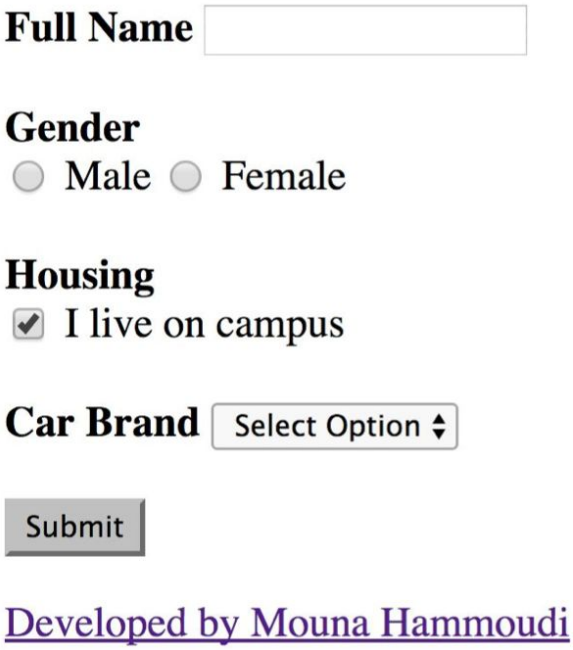

Figure 2.1: Student car registration form

script. Each Selenese command is denoted by a tuple: <action, locator, value>.[1] The action component of a Selenese command indicates either an event that is performed on the user interface (e.g., click, select) during the recording process, or an action specific to Selenium's control of the replay process. The locator component specifies the interface element (input field, dropdown list, etc.) that the user is interacting with during that step of the recording process. The value component refers to any input entered by the user within the specified locator, such as a value selected by the user within a dropdown list or a value typed in a text field.

Table 2.1: A Selenium Test

#	**Action**	**Locator**	**Value**
1	open	/~mouna/example.html	
2	click	css=*:checked	
3	click	document.sForm.elements[2]	
4	select	document.sForm.cars	label=Volvo
5	click	css=form#sForm>div> input[type="submit"]	
6	assertAlert	Please Fill All Required Fields	
7	type	id=FullName	John Smith
8	click	//html/body/form/div/input[5]	
9	clickAndWait	link=Developed by M.Hammoudi	

[1] Selenium itself uses the terms "command", "target" and "value"; we choose to use other terminology for clarity and generality.

```
<!DOCTYPE html>
<html>
  <head>
    <title> Student Information Form</title>
    <script type="text/javascript">
       function validateForm() {
           var FullName = document.forms["sForm"]["FullName"].value;
           if (FullName == null || FullName == "") {
               alert("Please Fill All Required Fields");
               return false;
           } else {
               document.getElementById("myForm").submit();
           }
       }
    </script>
  </head>
  <body>
    <form id="sForm" name="sForm" action="Form2.html" onsubmit="return validateForm()" method="
        post">
      <div class="Info">
        <b>Full Name</b>
        <input type="text" name="FullName" id="FullName" class="student">
        <br>
        <br>
        <b>Housing</b>
        <input type="checkbox" name="Living" value="Living" checked="checked"> I live on campus
        <br>
        <br>
        <b>Gender</b>
        <input type="radio" name="sex" value="male"> Male
        <input type="radio" name="sex" value="female"> Female
        <br>
        <br>
        <b>Car Brand</b>
        <select name="cars">
          <option value="Mercedes">Select Option</option>
          <option value="Volvo">Volvo</option>
          <option value="Saab">Saab</option>
          <option value="Fiat">Fiat</option>
          <option value="Audi">Audi</option>
        </select>
        <br>
        <br>
        <input type="submit" value="Submit" alt="Submit" onclick="validateForm()">
      </div>
    </form>
    <div>
      <p><a href="http://cse.unl.edu/~mouna"> Developed by M.Hammoudi </a>
      </p>
    </div>
  </body>
</html>
```

Figure 2.2: HTML for the student car registration form in Figure 2.1

2.2 HTML Elements, Attributes, and Text

HTML pages are composed of *HTML elements* such as text fields, radio buttons, checkboxes, dropdown lists, and links. HTML elements can have attributes that are known as *name/value pairs* (e.g., "id=FullName"), that provide data on the elements and allow developers to customize them. Attributes can do many things: among these, they can provide an identifier or name for an element, or specify visual characteristics of an element such as its width, height, or color. Attributes can also specify the "dynamic status" of an object; for instance, whether it should be enabled or disabled, or whether a checkbox should be checked by default.

We use the term "element text" to refer to the HTML textual content corresponding to an element that is visible in the UI of a web application. For instance, in Figure 2.1, an example of element text is the link text visible at the bottom of the page, "Developed by M.Hammoudi".

2.3 Locators

Locators are used by JavaScript and other languages, and by record/replay tools, to identify and manipulate elements. We identify two classes of locators, the second of which is composed of two sub-classes.

Attribute-based locators make use of element attributes such as element ids and names to identify elements, or reference element text in order to do so. For instance, in Table 2.1, Line 7 uses an element id and Line 9 uses element text.

Structure-based locators rely on the structure of a web page to identify elements. Among these, *hierarchy-based locators* specify elements in terms of their position in the DOM tree, using strategies involving xpaths, relative xpaths, or CSS selectors. For

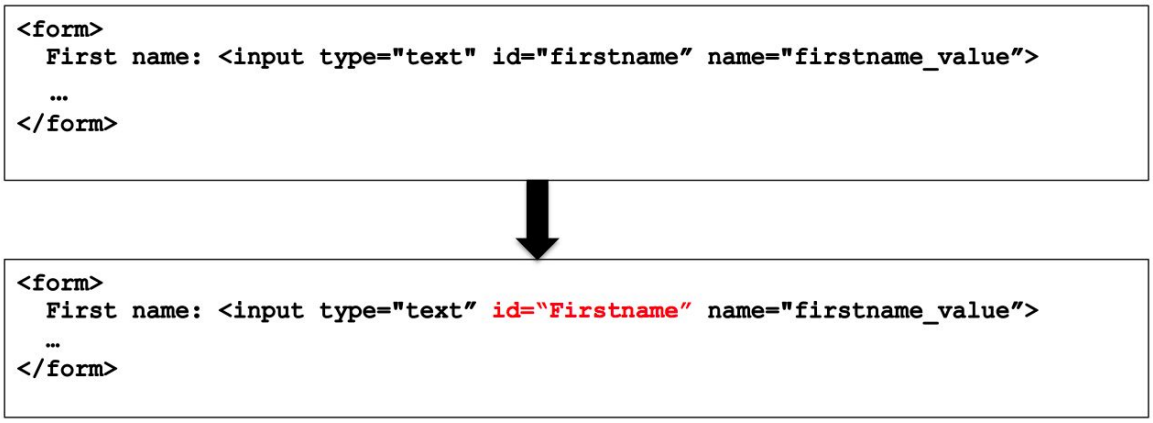

Figure 2.3: HTML code for version V (Upper Figure) and HTML code for evolved version V' (Lower Figure)

instance, in Table 2.1, Lines 2, 3, 4, 5, and 8 use structure-based locators. *Index-based locators* operate in situations where multiple elements in a web page have locators that are identical under other strategies; the indices differentiate them. For instance, in Table 2.1, Lines 3 and 8 use index-based locators. As Lines 3 and 8 show, the two structural locator types can appear together.

Figure 2.3 is an example of a locator breakage. The upper part of figure 2.3 represents the HTML code corresponding to version V of the web application under test and the lower part of figure 2.3 represents the HTML code corresponding to the evolved version V' of the web application. In this case, the ID of the "First Name" web element is capitalized in version V'. Let us consider a test that contains the following three-tuple structure in order to input the value "John" within the first name text field: <type, id="firstname", "John">. This statement would break at the level of its locator component given that the first name ID attribute value has been capitalized when moving to version V'. In this case, the proximal cause is that the locator of the "First Name" text field cannot be found and the distal cause is that the "First Name" text field ID has been capitalized when moving to version V' of the web application under test.

Chapter 3

Empirical Process

Our overall strategy for constructing a taxonomy was data-driven and bottom up. We gathered releases/commits of five web applications, used a record/replay tool to create tests for these, and applied these tests, recording all breakages. We used these results to create our taxonomy, and then we repeated the process on three additional web applications to provide an initial assessment of the taxonomy's validity and generality. We now describe the processes we followed to do this.

3.1 Objects of Study

To create and assess our taxonomy we searched for complex modern web applications being used by numerous people and under active development. Our criteria for selecting applications required that they: (1) have at least 20 releases/commits, (2) involve at least 30,000 lines of code, (3) be installable and executable, and (4) have been downloaded at least 5,000 times.

Table 3.1 provides data on the web applications that we selected, including their names, the number of releases/commits (hereafter usually referred to as "versions")

Table 3.1: Objects of Study

App Name	Vs.	LOC	Downloads	Tests
PHPFusion	49	256,899	1,605,195	47
Address Book	79	35,675	126,146	44
PHPAgenda	32	43,831	64,605	42
MyCollaboration	29	116,345	7,638	39
Joomla	111	312,978	>50,000,000	56
MyMovieLibrary	23	31,324	6,201	36
Dolibarr	30	42,010	864,698	38
YourContacts	100	64,765	676,543	57

we used (discussed further below), the number of lines of code (counted using `cloc`[1] and averaged across the versions), the number of downloads listed for them as of the day we selected them, and the number of tests used for them (also discussed later in this chapter[2]). The first five applications listed in Table 3.1 are the ones we used initially to develop the taxonomy (*our training set*), while the last three are the ones we utilized after the initial creation of the taxonomy to validate the applicability of the taxonomy on a different set of web applications (*our test set*).

`AddressBook` is a PHP web application that allows users to manage contacts and organize them based on their addresses, e-mails, phone numbers and birthdays. The application also allows users to download/upload Excel spreadsheets that contain the user's contacts. `AddressBook` makes use of various other technologies such as Google Maps, Gmail, Excel, LDIF and vCards, and uses a MySQL database to store all of the contacts.

`PHPFusion` is a web content management system that allows end users to create, manage and administer a website without requiring knowledge about web programming. `PHPFusion` supports news, articles, forums, photo gallery, web links, downloads, polls,

[1] cloc.sourceforge.net

[2] For a list of the releases/commits we considered, as well as pointers to the source code repositories for each object of study, see https://sites.google.com/site/taxonomyobjects.

shoutbox, themes and search. PHPFusion is widely used, and is available in more than 35 languages and has 21 support sites.

MyCollaboration is a web application based on JSP and servlets that is used as a collaboration platform to help users manage customer information and projects. It offers various features to users including project management, customer management, document management, sales management, file management, issue tracking, task management, user management, reporting, online editors and Gantt charts.

PHPAgenda is a PHP web application that allows users to manage calendars, schedule appointments, holidays, and todo lists, and share content with other users. It also allows users to regulate access to the web application and limit access to a specific category of users.

Joomla is an open source content management system that allows end users with little experience in web programming to publish content. Joomla makes use of the model-view-controller framework. Joomla offers a variety of functionalities to the user including the use of multiple languages, easy upgrades, an integrated help system, media management, banner management, contact management, search utilities, content management, content organization, tagging, version control, news management, the setting up of user access rights and cloud storage.

Dolibarr is a PHP web application used by companies for Enterprise Resource Planning and Customer Relationship management. Dolibarr includes a variety of features such as sales management, purchase management, event management, contact management, products and services catalogs, banking management, reporting, surveys, and project management.

YourContacts is a PHP application used by small and large companies to manage their business. Among the features included in YourContacts, we find contact

management, product management, invoice management, stock management, and employee management.

`MyMovieLibrary` is a Python web application used and keep track of the movies available in the store. `MyMovieLibrary` includes various functionalities for the store's management such as new movie purchases, new movie arrivals, and accounting. `MyMovieLibrary` also includes functionalities for customer management including new customer account management, customer charge management, customer account management, and customer gift card management.

`MyCollaboration` is written in Java, `Dolibarr` in Python, and the other applications are written in PHP. All applications use JavaScript, HTML, MySQL, and CSS.

3.2 Choice of Record/Replay Tool

To conduct our study we needed to choose a representative record/replay tool. As noted in Chapter I there are many such tools available. Ultimately we chose Selenium IDE, given its popularity, its open-source nature (which will support future work), and the features it supports.

Selenium IDE is in fact similar to many other record/replay tools in terms of the types of information and processes it uses to support replay. Other record/replay tools such as Watir [12], CasperJS [1], and Sahi [6] have test structures that utilize <action, locator, value> tuples and utilize similar types of locators. The fact that these tools use different syntaxes to encode test steps is immaterial where test breakages are concerned. Differences in the frequencies at which various locator strategies are employed, however, may indeed cause breakage data to differ. Neverthless, we expect

our taxonomy to generalize to other Record/Replay tools in this class, possibly with different breakage frequencies.

3.3 Selection of Versions/Commits

To choose releases for our objects of study, we relied on both commits and releases of web applications. The first five web applications listed in Table 3.1 were considered at the level of releases, whereas the last three web applications (Dolibarr, YourContacts and MyMovieLibrary) were considered at the level of releases and intermediate commits. In this latter case, we considered n equally spaced commits between successive pairs of web application releases. We chose both releases and commits in a way that they clearly represented checkpoints at which a given application could be deemed ready for deployment and testing.

We began by considering all versions of each web application, but found that we did have to exclude some. These included early versions of applications that were unstable or no longer functioned on current platforms. We also excluded releases/commits that contained serious faults. The numbers of versions listed in Table 3.1 are the numbers we ultimately retained.

3.4 Test Creation and Execution Process

In searching for web applications, we discovered that few open-source applications are provided with capture-replay test suites; in fact, few are provided with any test suites at all. In the case of the applications we selected, `Joomla` has a suite of WebDriver tests, but only for its first version, and Dolibarr has a single WebDriver test. However, we have no knowledge as to whether the tests were created through a capture-replay

approach, so using them was not a viable option. `PHPAddressBook` has unit tests that do not test the application's GUI. The other applications have no test suites. For this reason, we found it necessary to create new test suites.

To do this, we followed a systematic and iterative procedure for each web application. First, for each application the author installed its initial version, V_0, and familiarized herself with all of the functionalities provided by the application. Next, she constructed a test suite for V_0. To minimize the subjectivity involved in this task, we adopted functional adequacy and partition/boundary value coverage criteria. To achieve functional adequacy, the author created end-to-end use cases for the application, and continued to do so until all input fields and clickable items in each application had been exercised at least once. To achieve partition/boundary value coverage, she explicitly included in these use cases inputs exercising both nominal and exceptional behaviors, e.g., by choosing inputs, actions and boundary values that exercise non-mainstream application behavior, and by adding pauses to the input stream at appropriate moments to reflect user inactivity. Such use-case-based approaches to testing systems at the interface level are quite common in practice, and also well-suited to the use of record/replay tools.

Given a test suite T created for a version V (V_0 or a subsequent version) of a web application, the author executed that suite on the next version of the application, V', and noted each instance in which a test broke. She then manually repaired each of the tests that were repairable – an iterative process because repairing one test breakage might allow that test to proceed further and break again later. She also added new tests covering new functionality to the test suite. This resulted in a new test suite T' that now functioned on V'. The numbers of tests listed in Table 3.1 are the numbers of tests available on the final version of each object of study.

The foregoing process was repeated for each version of the web application until

each had been tested and the causes of all test breakages had been noted. (Again, this was applied in two phases, using the first five applications prior to taxonomy creation and the last three applications after it.) While following this process to repair and augment the tests for a given version V, no attempt was made to inspect the next version V' until after the repair and augmentation process was complete, reducing the threats to validity that might result if tests explicitly "targeted" particular aspects of subsequent releases.

As one further note, all of the objects of study made use of databases, and in order to test them these databases needed to be populated with initial data; this was done by the author initially with the first version, and then, as application modifications necessitated, new versions of the databases were created appropriate to new application versions/commits.

3.5 Taxonomization Process

After enumerating all causes of test breakages across the first five versions/commits of our object web applications, we began the process of creating a taxonomy. To do this we adopted a systematic process. Essentially, this process involved clustering causes of test breakages in terms of similarity factors. For each test breakage that had been encountered, the author studied it to determine its cause and wrote a description of it. Reviewing these descriptions allowed her to begin to identify candidate equivalence classes of test breakages, to which she assigned descriptive labels. These equivalence classes and labels were then reviewed with a colleague and the two reached a consensus on them. The author then collaborated with a colleague to organize the labels into hierarchies, again by using clustering based on the similarity of the factors responsible for the breakages.

For example, in version 7.00.06 of `PHPFusion`, a test uses a locator id "user_name" to locate an element (a text field to fill in). In the code of version 7.00.07, this id is changed to "AdminUsername", rendering the test unable to locate the text field. The description of this breakage was: *The test broke when a locator attempted and failed to locate an element based on its attribute, "user_name", because that attribute had been changed to "AdminUsername".* The label for this was "Locator break due to failure to locate element attribute". Later, review of these labels and consideration of other labels led us to rename this category of event "Element attribute not found", to instantiate "Attribute based locators" as a parent category of this type of change, and to instantiate "Locators" as a parent of this.

Taxonomies such as ours are conceptual maps derived from empirical observations; as such they typically evolve as additional observations of the world are made. We expect the same to be true of our taxonomy, and thus, it is not necessarily the case that any attempt to apply the taxonomy to subsequent applications will allow every type of test breakage in those applications to be categorized. In such cases the taxonomy will require adjustments. That said, we believe that a useful taxonomy should handle the majority of new situations it encounters. This is why we conducted the second (testing) phase of data collection and taxonomy validation.

3.6 Threats to Validity

External validity threats concern the generalization of our findings. We considered only eight web applications and our results may not generalize to different ones. The selection criteria that we adopted, however, did ensure that these applications were of non-trivial size and had multiple releases, and had experienced many downloads.

Moreover, as we shall see, our validation phase provides at least initial evidence that the taxonomy is more widely applicable.

As a second threat, our results are gathered relative to tests created and executed using the Selenium IDE record/replay tool. As such, generalization to programmable (e.g., Selenium WebDriver) and visual (e.g., Sikuli) tools is not supported by our current results. Still, as noted earlier, many other record/replay tools do operate in manners similar to Selenium IDE, so we conjecture that the use of such tools might lead to similar results. Nevertheless, the only way to definitively address this threat is by further study.

As a third threat, it might also be argued that developers who do create tests will behave differently when modifying their applications, resulting in different frequencies of breakages in certain taxonomic categories. This too requires further study, but we do return to it in Chapter VI when discussing breakage avoidance.

Internal validity threats concern uncontrolled factors that may have affected the construction of the taxonomy. The author created the tests. To control and reduce the subjectivity of this process, we defined precise adequacy criteria for test input creation and a precise procedure for test scenario construction; still, different tests might have revealed different breakages. The author also classified the test breakages, participating with a colleague. This task, however, requires reasoning that cannot be automated, so it is difficult to envision less threat-prone approaches. Moreover, to reduce the subjectivity involved in the task, she followed a systematic and structured procedure (write short breakage descriptions; assign labels to the descriptions; organize labels hierarchically), and interacted with a colleague on the task.

Construct validity threats concern our metrics and measures. We have measured the numbers of test breakages observed in each taxonomic category, but this may not

correlate with other important factors such as the difficulty of repairing or preventing (automatically or not) particular classes of breakages.

Chapter 4

Taxonomy of Causes of Test Breakages

Before presenting our taxonomy, we comment on two issues: (a) the types of breakage causes being considered; and, (b) the granularity of breakages. Throughout this chapter, let V be a version of a web application, let t be a test created and executed on V, and let V' be a new version of V.

4.1 Breakage causes

Our taxonomy focuses on the *proximal* causes of test breakages; that is, the causes "most closely related to" each breakage that are found in the test code. Proximal causes are usually related to *distal* causes, such as the changes that were made to the web application code that resulted in the breakage. For example, a web application developer may modify the code of V by removing a particular element (e.g., a choice in a dropdown list) from that page. This modification may cause t to break in V' when it attempts to access the element. The fact that the locator corresponding to

the dropdown list fails to point to any element is the proximal cause of the breakage; the fact that the developer modified the dropdown list is a distal cause.

As a second example, a developer may add a new page to V as the target of a `submit` button, such that t now reaches that page in V', and is unable to proceed further due to the inability to locate some element e in that new page. In this case the addition of the page is a distal cause, and the proximal cause is the failure of t to locate e.

Since we are interested in the evolution of test code, proximal causes are more relevant than distal causes. Thus, our taxonomy classifies the causes of test breakage that most nearly impact the test code and does not attempt to categorize the types of changes associated with the evolution of the web application code. That said, when presenting examples of concrete test breakages corresponding to the categories in our taxonomy, we often find it convenient to mention distal causes as well.

4.2 Granularity of Breakages

Tests can break multiple times, and we consider separately each distinguishable element of a test that acts as a proximal cause of a breakage. We have already noted in Chapter 3 that correcting one breakage and rerunning the test allows other breakages to appear later, and we count and categorize each such breakage. In addition, single statements or components of those statements can contain multiple causes of breakages. For example, suppose that V provides functionality allowing a user to change their password, via a text box having the id "password" in which the user enters their new password. Suppose that t contains the statement `type id=‘‘password’’ mypassword`, which locates the "password" text box and types "mypassword" into it. Suppose that in V', the id for the text box is capitalized ("Password") and additional validation

logic is added to the application code to verify whether the password is sufficiently complex (these modifications are distal causes). There are two separate breakages that may occur in this case: (1) t will break when it cannot locate the element attribute `id=‘‘password’’`, and (2) after that problem is corrected, t may break again if the password it enters is now invalid. Each of these breakages is a unique proximal cause that must be addressed separately, and we would count and categorize each of these individually.

A second example is worth considering. Locators are complex, and as noted in Section 2.3, different locator types can be combined. For example, the locator `//span[@name=‘‘pass-info’’]/div/button[0]` points to a button in a web page; it contains an attribute-based locator component (“pass-info”) and makes use of both hierarchy-based and index-based components. Suppose this locator is present in t, and suppose that in V' three changes (distal causes) are made: (1) “pass-info” is changed to “passenger-info”, (2) the DOM tree changes such that the “/div” tag is deleted, and (3) the button is repositioned within the web page such that it is no longer in position 0. In this case, t will break when it tries to use this locator, but the breakage is due to three unique proximal causes, including the failure to locate an element attribute, the failure to locate a hierarchy-based locator target, and the failure to locate an index-based locator target. To correct t, each of these three proximal causes must be addressed. We thus count and categorize each of these separately.

Thus, to summarize, the data we present in this thesis on test breakages pertain to proximal causes, and are collected with respect to all individual (“atomic” if you will) discrete proximal causes, whether these are located in a single Senelese statement or statement component, or not.

Figure 4.1 presents a graphical view of our taxonomy of causes of test breakages. Nodes represent classes and subclasses of breakages, and edges represent complete,

disjoint specialization relationships (i.e., given a node A that has edges to nodes B and C, A is a class of breakages that consists of two mutually exclusive subclasses, B and C). The numbers in non-root nodes uniquely identify each node while also identifying their chain of ancestors (if any). The numbers on edges are described in Chapter 5.

At the highest level of our taxonomy we identified five distinct categories of causes of test breakages, as follows:

1. Causes related to locators used in tests.
2. Causes related to values and actions used in tests.
3. Causes related to page reloading.
4. Causes related to changes in user session times.
5. Causes related to popup boxes.

Each node within our taxonomy is related to one or more components within the three-tuple structure of a Selenese command <command, target, value> presented in Section 2.1. The first node of our taxonomy is related to locator breakages. Locators are specified through the target component of a Selenese command. The second node of our taxonomy represents value and action breakages, which are also related to the value and action components of a Selenese command. The third node of our taxonomy accounts for page reloads. Page reloads are specified through the action component of a Selenese command (the action ClickAndWait is used in the case of a page reload, and the action Click is used in cases in which a page is not reloaded). The fourth node of our taxonomy is specific to user sessions and is related to two components within a Selenense command: action and value. Finally, the last node of our taxonomy

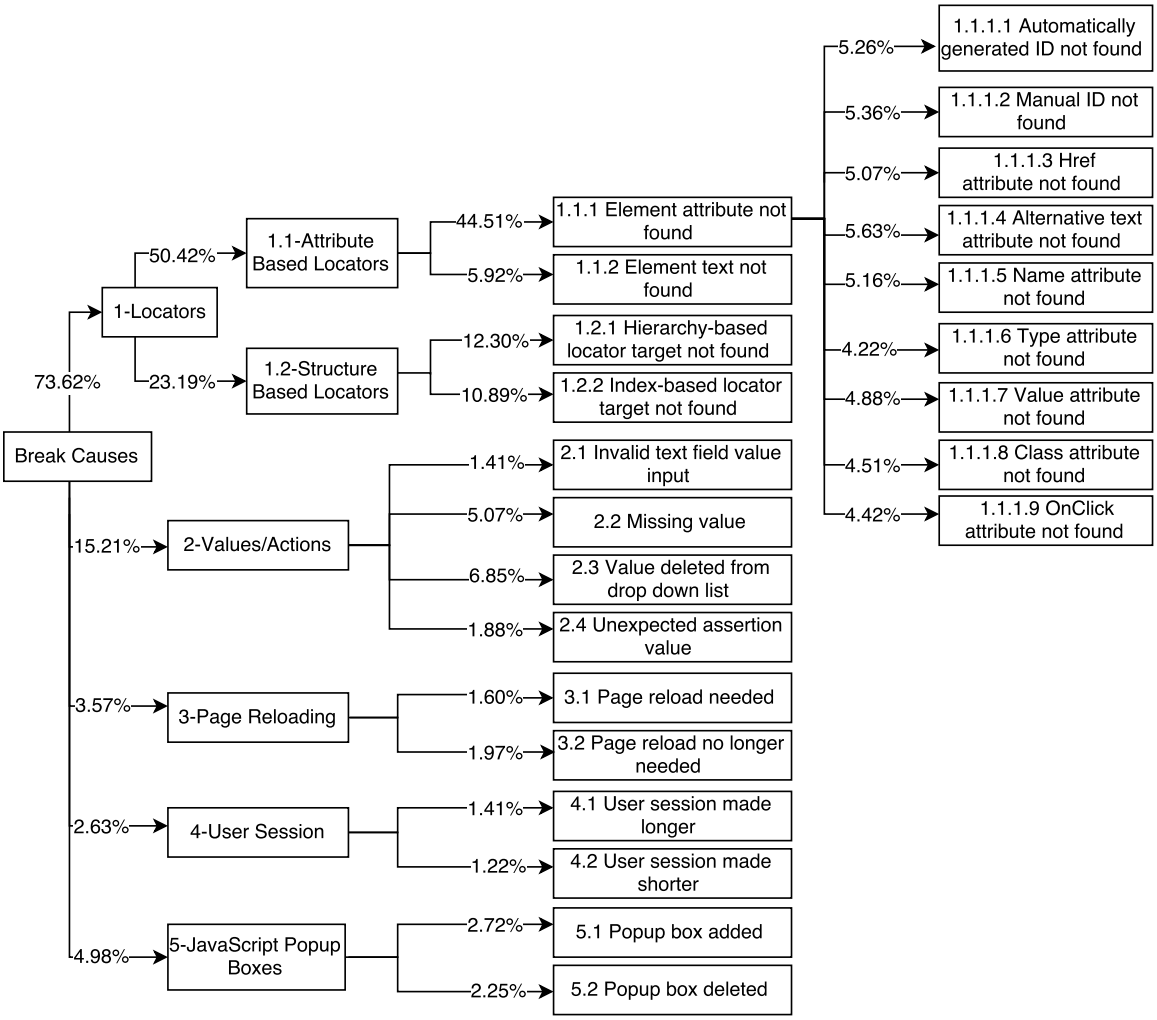

Figure 4.1: Taxonomy

is specific to popup boxes. The occurrence of popup boxes is represented through a Selenese statement specifying an action and a value. The action component represents the occurrence of the popup box (the AssertAlert action) and the value component accounts for the message displayed within the popup box.

We now discuss each of these categories in turn, in each case considering all subcategories beneath them.

4.3 Locators (Category 1)

The test breakage causes in this category are all proximally related to locators. Overall, this category has two subcategories, corresponding to the two major classes of locators described in Chapter 2: attribute-based and structure-based.

4.3.1 Attribute-Based Locators (Category 1.1)

As noted in Chapter 2, elements of web pages have attributes and record/replay tests can make use of these to locate elements. They do this by identifying an element using its attribute's name/value pair or by looking for the element text, i.e., the HTML content that is visible in the user interface. We consider each of these access approaches separately.

4.3.1.1 Element Attribute not Found (Category 1.1.1)

This breakage is caused when t attempts to locate an element e via an attribute a that functioned (led to e) in V, but no longer functions in V'. The "Element Attribute not Found" subcategory can be divided into multiple subcategories depending on the type of attribute causing the test to break. We present these next.

Automatically Generated ID not Found (Category 1.1.1.1). This breakage is caused when t attempts to locate an element e via an automatically generated ID id that functioned (led to e) in V, but no longer functions in V'. There are many potential distal causes of such a breakage. For example, the test presented in Table 2.1 may break at Line 7 when attempting to locate the "Full Name" text field in the application if the developer uses a numerical ID for the full name text field and if the ID of the full name text field is automatically incremented whenever a new submission of the car registration form presented in Figure 2.2 is made. Another distal cause that would

cause a test breakage in Line 7 of the test case involves a scenario in which a random ID is automatically generated for the full name text field whenever a new submission of the form is made.

Manual ID not Found (Category 1.1.1.2). This breakage is caused when t attempts to locate an element e via a manual ID id that functioned (led to e) in V, but no longer functions in V'. There are many potential distal causes of such a breakage. For example, the test presented in Table 2.1 may break at Line 7 when attempting to locate the "Full Name" text field in the application if Line 21 of the HTML code shown in Figure 2.2 is altered such that `id=''FullName''` is now `id=''Name''`. Another distal cause that would lead to such a breakage could be related to the deletion of Line 21 of the HTML code of the web application under test. This would lead to the deletion of the $FullName$ web element from the DOM tree of the web application under test, and this would eventually cause the test represented in Table 2.1 to break at Line 7.

Href Attribute not Found (Category 1.1.1.3). This breakage is caused when t attempts to locate a link l via an $href$ attribute value that functioned (led to l) in V, but no longer functions in V'. There are many potential distal causes of such a breakage. For example, the test presented in Table 2.1 may break at Line 9 when attempting to locate the "Developed by M. Hammoudi" link in the application if Line 47 of the HTML code shown in Figure 2.2 is altered such that `href=''http://cse.unl.edu/ mouna''` is now `href=''http://cse.unl.edu/ mounahammoudi''`. Another distal cause that would lead to such a breakage could be related to the deletion of Line 47 of the HTML code of the web application under test. This would lead to the deletion of the `''Developed by M. Hammoudi''` link from the DOM tree of the web application

under test, which would eventually cause the test represented in Table 2.1 to break at Line 9.

Alternative Text Attribute not Found (Category 1.1.1.4). This breakage is caused by an obsolete alternative text attribute. The *alt* attribute is used to define an alternative text string to be displayed in a scenario in which an HTML web element cannot be shown on the user interface. The "alternative text not found" breakage would be caused in a situation when t attempts to locate a web element e via an *alt* attribute value that functioned (led to e) in V, but no longer functions in V'. There are many potential distal causes of such a breakage. For example, the test presented in Table 2.1 may break at Line 5 when attempting to click on the "Submit" button in the application if Line 43 of the HTML code shown in Figure 2.2 is altered such that `alt="Submit"` is now `alt="SubmitButton"`. Another distal cause that would lead to such a breakage could be related to the deletion of Line 43 of the HTML code of the web application under test. This would lead to the deletion of the "Submit" button from the DOM tree of the web application under test, which would eventually cause the test represented in Table 2.1 to break at Line 5.

Name Attribute not Found (Category 1.1.1.5). This breakage would arise in a situation when t attempts to locate a web element e via a *name* attribute value that functioned (led to e) in V, but no longer functions in V'. There are many potential distal causes of such a breakage. For example, assume that the attribute `name="sex"` is used in Line 3 of the test case presented in Table 2.1 and let us suppose that the name attribute value in Line 29 of the HTML code shown in Figure 2.2 is changed from `name="sex"` to `name="gender"`. In this scenario, the test represented in Table 2.1 would break at Line 3 when attempting to click on the "male" radio button. Also, the deletion of Line 29 of the HTML code represented in Figure 2.2 would cause the

test presented in Table 2.1 to break in Line 3, due to the deletion of the "male" radio button from the DOM tree of the web application under test.

Value Attribute not Found (Category 1.1.1.6). This breakage would arise in a situation when t attempts to locate a web element e via a *value* attribute that functioned (led to e) in V, but no longer functions in V'. There are many potential distal causes of such a breakage. For example, assume that the attribute `//input[@value='male']` is used in Line 3 of the test case presented in Table 2.1 and suppose that the value attribute value in Line 29 of the HTML code shown in Figure 2.2 is changed from `value="male"` to `value="female"`. In this scenario, the test represented in Table 2.1 would break at Line 3 when attempting to click on the "male" radio button. Also, the deletion of Line 29 of the HTML code represented in Figure 2.2 would cause the test presented in Table 2.1 to break in Line 3, due to the deletion of the "male" radio button from the DOM tree of the web application under test.

Class Attribute not Found (Category 1.1.1.7). Class attributes are used to assign a CSS style to one or more web elements within the user interface of the web application under test. This breakage would occur in a situation when t attempts to locate a web element e via a *class* attribute that functioned (led to e) in V, but no longer functions in V'. There are many potential distal causes of such a breakage. For example, assume that the XPath `//div[@class="Info"]/input[1]` is used in Line 7 of the test case presented in Table 2.1 and suppose that the class attribute value in Line 19 of the HTML code shown in Figure 2.2 is changed from `class="Info"` to `class="PersonalInfo"`. In this scenario, the test represented in Table 2.1 would break at Line 7 when attempting to identify the "FullName" text field. Another

situation in which the test would break in Line 7 would be caused by the deletion of the class attribute from Line 19 of the HTML code shown in Figure 2.2.

OnClick Attribute not Found (Category 1.1.1.8). The *onClick* attribute leads to the execution of a JavaScript function whenever a button is clicked. The onClick attribute value specifies the name of the JavaScript function to be executed. This "OnClick Attribute not Found" would occur in a situation when t attempts to locate a web element e via its *onClick* attribute value that functioned (led to e) in V, but no longer functions in V'. There are many potential distal causes of such a breakage. For example, assume that the XPath `//input[@onclick="validateForm()"]` is used in Line 8 of the test case presented in Table 2.1 and suppose that the JavaScript function $validateForm()$ is renamed into $validateSubmission()$ in Line 6 of the HTML code shown in Figure 2.2. In this scenario, the test represented in Table 2.1 would break at Line 8 when attempting to identify the submit button due to an obsolete *onClick* attribute value. Furthermore, suppose that the web application evolves such that no JavaScript function needs to be executed anymore after clicking on the *submit* button. This would be a scenario where the $validateForm()$ JavaScript function is deleted from the HTML code shown in Figure 2.2. In this case, the deletion of the $validateForm()$ JavaScript function would constitute a distal cause of the failure to locate the submit button based on the value of its OnClick Attribute.

4.3.1.2 Element Text not Found (Category 1.1.2)

This breakage is caused when t attempts to locate an element e via its text, using a text string that functioned correctly (led to e) in V but no longer functions in V'. There are many potential distal causes of such a breakage. For example, the text in

the element may have been modified. As a concrete instance, the test presented in Table 2.1 will break at Line 9 if the link text “Developed by M.Hammoudi” is changed.

4.3.2 Structure-Based Locators (Category 1.2)

As noted in Chapter 2, structure-based locators can be either hierarchy-based (utilizing the position of elements in the DOM tree), or index-based (utilizing indices), or both, and we distinguish between these two cases.

4.3.2.1 Hierarchy-Based Locator Target not Found(Category 1.2.1)

This breakage is caused when t attempts to locate an element e via a hierarchy-based locator l that functioned (led to e) in V, but no longer functions in V'. Distal causes of this breakage include the addition, deletion or modification of an ancestor of e in the DOM tree. For example, suppose that the “div” ancestor of the “Submit” button in the DOM tree for the application shown in Figure 2.1 is deleted (by deletion of Lines 19 and 44 in the code). In this case the test presented in Table 2.1 will break at Lines 0 and (after partial repair) at Line 8.

4.3.2.2 Index-Based Locator Target not Found (Category 1.2.2)

This breakage is caused when t attempts to locate an element e via an index-based locator l that functioned (led to e) in V, but no longer functions in V'. Distal causes of this breakage include the addition or deletion of elements to/from the web page that satisfy the same locator strategy of an element in V, but cause a change in the index of the element that t was intended to select. For example, suppose the text field for "Full Name” is removed from the application shown in Figure 2.1. In this case the test presented in Table 2.1 will erroneously select the “Female” radio button at Line 2

and toggle that instead of the "Male" radio button, which is not the test's intended behavior. The test will then break upon reaching the `assertAlert` at Line 6, because the event that triggers that alert (absence of input to the "Full Name" text field) no longer occurs.

4.3.3 Values (Category 2)

All of the causes of breakages in this category are proximally related to values within tests.

4.3.3.1 Invalid Text Field Input (Category 2.1)

This breakage is caused when values used by t as inputs to V are no longer accepted in V'. For instance, if t inputs a password in V, and in V', restrictions on passwords are implemented that weren't present in V (e.g., required characters) then the password input by t may now be invalid on V', in which case t will break.

4.3.3.2 Missing Value (Category 2.2)

This breakage is caused when values or actions not previously included in t as inputs to V are required in V'. For instance, a new text field or checkbox may be added to a form in V', such that the form cannot be submitted unless an input is provided to the text field or an action is applied to the checkbox. As another example, this breakage can occur if a field or other page element for which inputs or actions are optional in V, and omitted from t, becomes required in V'.

4.3.3.3 Value Absent from Dropdown List (Category 2.3)

This breakage is caused when a value that was previously selected by t from a dropdown list in V is no longer present in V'. For instance, a test of the web application presented in Figure 2.1 may select the option "Fiat" from the "Car Brand" dropdown list; if that option is missing in a succeeding version of that application, the test will break.

4.3.3.4 Unexpected Assertion Value (Category 2.4)

Assertions in record/replay tests can compare expected values to actual values, and signal cases in which these do not match. If the behavior of t with respect to an assertion differs when moving from V to V', then t will break. This can occur, for example, if the text of an error message raised by the assertion changes in V'.

4.3.4 Page Reloading (Category 3)

The breakages within this category are all proximally related to page reloading activities. Page reloading may be triggered by user actions when, say, the current web page is updated in response to a user event. For example, in Figure 2.1, if clicking on the "I live on campus" checkbox causes new fields such as "campus address" to be needed on the web page, a reload is needed. Like most record/replay tools, Selenese relies on a command (`clickAndWait`) to give the reload enough time to complete. There are two subcategories of these breakages.

4.3.4.1 Page Reload Needed (Category 3.1)

This breakage is caused if t, run on V, does not trigger a page reload, but does trigger a page reload when run on V'. In such cases, t may break due to the lack of a delay sufficient to allow its next action to succeed. Such a case may also arise if an action

performed by t in V' causes the page to be reloaded in a longer time than was required in V, e.g., in order to present results of database queries that require longer wait times in the new version.

4.3.4.2 Page Reload no Longer Needed (Category 3.2)

This breakage is caused if t, run on V, does trigger a page reload, but does not trigger a page reload when run on V'. In such cases, t will attempt to wait for a page reload that will not occur, and will time out with an error message. Such a case could arise, for example, if Ajax facilities were introduced in V' to allow a partial page update in place of a full reload.

4.3.5 User Session Times (Category 4)

Many web applications utilize "timeouts" on user sessions; for example, a bank application may automatically log users out who are inactive for a certain amount of time. The breakages within this category are all proximally related to user session times. There are many ways in which timeouts may occur; for example, after m minutes of inactivity, a user may receive a popup message letting them know that they will be logged out in n minutes. User session limits may be encoded differently in tests depending on the manifestation of the session timeout. For instance, in the popup case, the test may contain statements that do the following: (1) wait for m minutes to simulate user inactivity, (2) issue an "assert Alert" announcing the impending logout. There are two subcategories of breakages related to user session times, as follows.

4.3.5.1 User Session Made Longer (Category 4.1)

This breakage is caused if t, run on V, is meant to test (and wait for) a user session timeout of s_1 seconds, but when run on V', it is required instead to test (and wait for) a user session timeout of s_2 seconds, where s_2 is sufficiently longer than s_1. These breakages may be manifested in various ways; for instance, an alert JavaScript popup box containing the message "you will be logged out" may fail to appear in the time specified by the wait command, and the test will break.

4.3.5.2 User Session Made Shorter (Category 4.2)

This breakage is caused if t, run on V, is meant to test (and waits for) a user session timeout of s_1 seconds, but when run on V', it is required instead to test (and wait for) a user session timeout of s_2 seconds, where s_2 is shorter than s_1. These breakages may be manifested in various ways; for instance, V' might automatically log the user out prior to the completion of the wait command, disabling any further actions by the test.

4.3.6 Popup Boxes (Category 5)

Popup boxes are often used in web applications to display error or other forms of messages. The breakages within this category are all proximally related to popups. Selenese relies on commands such as `assertAlert` to check whether popups function as expected. There are two subcategories of these breakages, as follows.

4.3.6.1 Popup Box Added (Category 5.1)

This breakage is caused if t, run on V, did not expect a particular occurrence of some popup box B, but when run on V' it does encounter B. In this case, the test breaks

due to the presence of the unexpected popup box. For example, in V', the act of clicking on a submit button may trigger the appearance of a popup box containing the text "Thank you for your submission" that was not present in V.

4.3.6.2 Popup Box Deleted (Category 5.2)

This breakage is caused if t, run on V, did expect a particular occurrence of some popup box B, but when run on V', it does not encounter B. In this case, the test breaks due to the absence of the expected popup box. An example of this is the inverse of the example described for Category 5.1.

Chapter 5

Results

Tables 5.1 and 5.2 list the numbers of different causes of test breakages observed per web application and overall, for each of the leaf nodes in the taxonomy. We include data on all eight applications in the tables, with the table headers differentiating between the web applications utilized in the taxonomy construction phase (training set), and the applications utilized in the taxonomy validation phase (test set). The two rightmost columns of the tables list the total numbers and percentages of causes of test breakages across all applications, per category, across all observed causes of breakages [1].

We have also annotated the taxonomy graph in Figure 4.1, adding percentages to the edges. A percentage of k% on edge ($N1$-$N2$) indicates that k% of the total number (1065) of causes of test breakages are associated with that edge. These annotations show data pertaining to both leaf and non-leaf nodes; for example they show that, across all causes of test breakages, 73.62% involved locators, 15.21% involved values and actions, 3.57% involved page reloading, 2.63% involved user sessions, and 4.98% involved popup boxes. The data on percentages of individual breakage categories show

[1] Tables 5.1 and 5.2, as noted, present overall results per web application. For results per version, see Appendix A

Table 5.1: Test Breakage Cause Counts per Taxonomy Category and Web Application (Category 1.1.1 Not Expanded)

	Training set					Test set				
Category	PHP Fusion	Address Book	PHP Agenda	MyCollab	Joomla	MyMovie Library	Dolibarr	Your Contacts	Total	%
1.1.1 Element attribute not found	45	63	31	53	138	38	52	54	474	44.51
1.1.2 Element text not found	10	7	3	8	15	4	9	7	63	5.92
1.2.1 Hierarchy -based locator target not found	33	16	14	16	11	12	14	15	131	12.30
1.2.2 Index-based locator target not found	39	6	12	11	11	13	7	17	116	10.89
2.1 Invalid text field value input	2	3	1	2	1	2	1	3	15	1.41
2.2 Missing value	4	5	6	3	18	5	4	9	54	5.07
2.3 Value deleted from drop down list	6	16	17	9	4	6	8	7	73	6.85
2.4 Unexpected assertion value	2	1	2	2	3	3	3	4	20	1.88
3.1 Page reload needed	2	1	1	1	2	2	4	4	17	1.60
3.2 Page reload no longer needed	2	2	1	1	6	2	2	5	21	1.97
4.1 User session made longer	3	1	3	1	1	1	3	2	15	1.41
4.2 User session made shorter	3	1	3	1	1	1	1	2	13	1.22
5.1 Popup box added	2	2	1	3	13	2	1	5	29	2.72
5.2 Popup box deleted	2	1	3	6	3	4	1	4	24	2.25
Total	155	125	98	117	227	95	110	138	1065	100.00

that a preponderance of breakages (44.51%) involved element attributes not found, with the two types of structure-based locators being next (though far less) prevalent at 12.30% and 10.89%, respectively.

Beyond locator problems, deletion of values from dropdown lists and missing values were the next most prevalent at 6.85% and 5.07%, respectively.

Table 5.2: Test Breakage Cause Counts per "Element Attribute not Found" Category and Web Application (Category 1.1 Expanded)

	Training set					Test set				
Category	PHP Fusion	Address Book	PHP Agenda	MyCollab	Joomla	MyMovie Library	Dolibarr	Your Contacts	Total	%
1.1.1.1 Automatically generated ID not found	11	15	10	10	21	5	5	9	56	5.26
1.1.1.2 Manual ID not found	5	8	5	5	15	6	6	6	57	5.36
1.1.1.3 Href Attribute not found	5	5	4	7	19	5	6	5	54	5.07
1.1.1.4 Alternative Text Attribute not found	6	4	3	3	22	4	4	7	60	5.63
1.1.1.5 Name Attribute not found	4	9	4	5	13	3	7	5	55	5.16
1.1.1.6 Type Attribute not found	5	6	3	3	8	2	6	5	45	4.22
1.1.1.7 Value Attribute not found	3	5	2	4	11	2	8	10	52	4.88
1.1.1.8 Class Attribute not found	2	4	0	7	14	7	6	3	48	4.51
1.1.1.9 OnClick Attribute not found	4	7	0	9	15	4	4	4	47	4.42
Total	45	63	31	53	138	38	52	54	474	44.51

The data gathered on the applications in our test set largely mirror those gathered on applications in our training set, and the current taxonomic categories we created sufficed to categorize breakages in the test set. In particular, there was no single test breakage in the applications used for validation that could not be assigned to a category in the taxonomy, and no single category associated with breakages that did not occur in the additional web applications.

Chapter 6

Discussion

Our taxonomy and associated data could be useful in several contexts related to test breakages.

6.1 Prioritization of Breakages for Test Repair

Our data suggests which categories of test breakages merit the greatest attention. Locators caused over 73% of the test breakages we observed, and attribute-based locators caused the majority of these. Clearly, addressing just this class of errors by finding ways to repair them if they break would have the largest overall impact on the reusability of tests across releases. The data also suggest where subsequent priorities should be placed in terms of finding methods for repairing test breakages.

6.2 Prioritization of Inspections

Knowledge about the causes of test breakages could help test developers manually repair tests in the absence of automated techniques. Given a broken test, they can

prioritize the inspection of possible causes based on the relative importance of such causes in our taxonomy (e.g., by inspecting locators first, values next, etc.).

6.3 Breakage Avoidance

One example of an avoidance-based approach to test breakages involves educating developers and maintainers of web applications as to the causes and probabilities of test breakages. Such education could be supported by information present in our taxonomy and the data that underlie it, which could serve as a source of guidelines on best practices to follow while changing web application code. For example, we observed that many of the breakages involving attribute-based locators resulted from simple name changes, i.e., changing an attribute from `name=‘‘submitbutton’’` to `name=‘‘SubmitButton’’`. While such name changes may be helpful for code readability, their cost in terms of impact on tests can be quite high.[1]

6.4 Breakage Prevention

Knowledge about causes of test breakages could also help web application and test developers prevent them. For instance, they may introduce the practice of adding meaningful and stable `id`'s to the core elements of the web interface, or they may take advantage of tools like ATA [17] or ROBULA [15] to create robust locators automatically. Another way to produce change-resilient tests is by adopting design

[1] In our discussion of threats to validity (Chapter 3.1.6) we noted that developers who use capture-replay tests may be well aware that they should avoid changing attribute locators since they are likely to know that this can break tests. We did find, however, that on `Joomla`, where WebDriver tests were provided, there was still a large number of test breakages related to changes in attribute locators, as shown in Table 5.1. Further, in organizations in which testing is performed by persons other than developers, the incentive to avoid changes may not be strong.

patterns such as the *Page Object*[2] pattern. A Page Object acts as a mediator between a test and the web page whose elements it references. Such a mediator hides the internal, implementation details of the web page DOM from the test code by exposing only abstract interaction functions. When a breakage occurs, only the affected Page Object needs to be fixed; all tests using it will be repaired at the same time. Approaches such as this, however, are not yet widespread. Our data could help motivate their adoption.

6.5 Bad Smells in Web Tests

Many techniques exist for statically analyzing code to locate "bad smells" [19] – stylistic problems or malfeasances that developers may wish to guard against. Guided by our taxonomy, analogous tools could analyze web applications and their tests for potential problems. For instance, analyses could detect the absence of stable anchors for DOM elements that are expected to be important during testing (e.g., a `div` tag, showing the results produced after a form submission that are potentially useful to create test assertions). Where test breakages are concerned, however, such analyses might more appropriately be directed at pairs of versions of web applications, because it is the changes made to applications that are the distal causes of breakages and need to be considered. For instance, a DOM element that caused repeated test breakages in the past might deserve a dedicated localization mechanism (e.g., a unique `id` or `name`).

6.6 IDE Enhancements

An "after-the-changes-are-made" approach to analysis may be valuable, but it seems likely that an analysis approach that operates concurrently with program maintenance

[2] https://code.google.com/p/selenium/wiki/PageObjects

might be even more effective, and such an analysis could also be guided by our taxonomy. Modern IDEs for code development typically employ such approaches: as programmers edit, they point out problems or provide useful information based on the programmers' actions. Web application development IDEs employ such approaches also, but to our knowledge, no such IDEs also build in assistance related to testing efforts. Such IDEs could aid in test breakage avoidance by alerting programmers to possible effects and letting them choose whether or not to act on them. They could aid in prevention by prohibiting certain actions or by recommending the creation of stable anchors for locators whenever they recognize that locators used in tests are likely to be fragile. They could maintain information on changes made to a web application, that testers could later use when reviewing tests in preparation for applying them to a new version (a notion also suggested by Daniel et al. [20]). Such information could also be utilized by automated test repair techniques themselves. In that context, instead of postponing test repair efforts until later, such IDEs might even be capable of programmatically correcting tests in response to particular edits by refactoring test code related to locators, values, and so forth as the code is edited.

6.7 Root Cause Analysis

A final issue pertains to identifying test breakages and locating (or helping engineers locate) the proximal causes of those breakages. As we classified the causes of test breakages based on actual data, it became quite clear that the time at which the proximal cause of a breakage occurs and the time at which a breakage is detected can differ widely. *Direct breakages* manifest themselves precisely when the breakage cause is encountered, as for example when a popup box is added or deleted (taxonomy categories 5.1 and 5.2). *Propagated breakages* do not manifest themselves immediately,

but do manifest themselves later on subsequent test actions. For example, a specific attribute value a may be used in two different elements e_1 and e_2 of web application V, and test t may use a as a locator. In a version V' of the web application, if the use of a in e_1 has been deleted, t will refer to the use of a in e_2 instead. Depending on the result of this action, t may continue for some time before encountering a situation in which its next action cannot be completed. *Silent breakages* never manifest themselves explicitly. The previous example functions here as well, if t remains able to continue through until completion. In this case, t can still be utilized as a test, but from a test design standpoint this is quite problematic because t is not now testing what it was designed to test. These examples suggest that in addition to preventing, avoiding, or repairing test breakages, we need better methods for detecting them and for tracing them back to the problematic test code that serves as their proximal cause. Methodologies that monitor test execution for problems, perhaps similar to those created to monitor web macro execution for problems [21] might be effective here.

Chapter 7

Related Work

There has been some research on automated techniques for repairing *programs* (e.g., [22, 23, 24, 25, 26, 27, 28]). Our work focuses on tests.

There have been numerous papers on test repair. Several papers have addressed the problem of repairing unit tests such as JUnit tests or tests written in similar frameworks (e.g., [20, 29, 30]). Such approaches, however, are not applicable to tests produced by means of record/replay tools.

Many papers have addressed the problem of repairing GUI tests. GUI tests often track sequences of user actions applied to an interface, and some of these approaches may be adaptable to web applications and record/replay tests. Memon and Soffa [31], and Memon [32] and Datchayani et al. [14] use event flow graphs and transformation techniques to repair broken GUI tests. Huang et al. [33] use a genetic algorithm to repair GUI tests. Grechanik et al. [34] present an approach that analyzes an initial and modified GUI for differences and generates a report for engineers documenting ways in which associated test scripts may be broken by changes. Daniel et al. [35] use GUI refactorings to keep track of changes engineers make to a GUI, suggesting that this information could later be used to repair tests.

Zhang et al. [36] address the problem of repairing broken *workflows* in GUI applications, where a workflow is a sequence of activities to perform a given task. This approach, however, does not target tests.

Alshawan and Harman [37] present an approach for repairing user session data used to regression test web applications. User session data, however, differs from the data captured by record/replay tools; it records just the requests received by a server from a web application.

Four recent papers consider problems related to record/replay tests of web applications. Stocco et al. [38] investigate the automated generation of page objects that confine causes of test breakages to a single class, a form of breakage prevention. They develop a technique that ensures the automatic creation of page objects to reduce the tester's effort in the context of web test maintenance. More specifically, Stocco et al. develop a technique that addresses the problem of test code duplication. Thanks to page objects, whenever changes are made to the web application under test, the tester needs to repair the breakage only once. Such a repair is automatically propagated to all tests using the broken element.

Yandrapally et al. [17] address the problem of test script fragility in relation to locators, proposing approaches for identifying UI elements in more robust ways using contextual clues, also a form of prevention. They create a new record/replay approach that identifies a web element based on other elements surrounding it. Hence, Yandrapally et al.'s technique automates the process of test case creation and prevents future test breakages by keeping track of contextual clues within the user interface of the web application under test.

Another approach for producing robust locators has been implemented in the tool ROBULA [15]. ROBULA is used for preventing test breakages and reducing web test case aging through the generation of robust XPath locators that are less likely to break

whenever the web application evolves. The multi-locator extension of ROBULA [16] supports automated repair of broken locators.

Choudhary et al. [13] focus, as do we, on test scripts such as those created by Selenium, and present WATER, a technique based on differential testing that compares the behavior of a broken test to its behavior on a prior version of the web application. Based on the differences found, the technique suggests a list of potential test repairs for each breakage that was encountered. WATER suggests test repairs relative to obsolete locators, obsolete assertions, newly added form elements and deleted form elements.

Researchers have presented various fault taxonomies. Bruning et al. [39] present a fault taxonomy for service-oriented architectures. Hayes [40] provides a fault taxonomy for NASA software requirements. Mariani [41] presents a taxonomy for component based software systems. Chan et al. [42] provide a taxonomy for web service compositions. Hummer et al. [43] present a taxonomy for event-based systems. None of these taxonomies consider web applications.

A few papers present fault taxonomies for web applications. Ocariza et al. [44, 45] characterize the classes of error messages output by JavaScript in web applications and the classes of faults found in JavaScript. Ricca and Tonella [46] present a preliminary fault taxonomy for web applications, consisting of a single level of general fault categories. Marchetto et al. [47] present a taxonomy of web application faults. None of these papers consider test breakages.

Leotta et al. [48] conducted an empirical study in which they measured the effort associated with the evolution of two equivalent Selenium IDE and Selenium WebDriver test suites, with the aim of comparing the maintainability of record/replay vs. programmable test suites. Changes made to the tests were classified as logical or structural. However, no further refinement of this classification was carried out.

To our knowledge, the only paper that provides a more detailed classification of the ways in which tests of web applications may break is the test repair paper by Choudhary et al. [13], described earlier. The authors identify three types of changes related to broken test scripts. (1) Structural changes are changes in the DOM tree and may cause locator problems that can be characterized as "non-selection" or "mis-selection" problems. One type of structural change involves additions, deletions or modifications of form elements and these comprise the "form data problem". (2) Content changes involve modifications of the text or HTML contained in a DOM node; these can affect assertions that check node contents, and lead to "obsolete content" problems. (3) Blind changes involve changes in server-side code. This classification focuses first on changes and then on associated problems that may result in test breakages, whereas we focus on breakages. Further, while based on the authors' experience, the classification is not drawn from any formal empirical study of web applications, whereas ours is. Finally, our taxonomy recognizes several types of breakages not noted by Choudhary et al., handling blind changes by focusing on proximal causes, which involve events or actions occurring on the client side.

Chapter 8

Conclusions and Future Work

Following a rigorous process based on empirical observation of hundreds of versions and/or commits of non-trivial, popular web applications, we have constructed a taxonomy of the causes of breakages of tests created by record/replay approaches for web applications, and the frequency at which the identified causes contribute to test breakages.

As future work, we would like to investigate the uses of our taxonomy for test breakage prevention via the creation of tests that are less brittle and less likely to break in the face of web application evolution. Furthermore, we would like to create novel repair techniques that would facilitate the automatic repair of broken tests. Also, we would like to create automated techniques for preventing test breakages by warning developers about the repercussions of changes they make on web applications. We are also interested in test breakage avoidance, and approaches that stop developers from making web application changes that would damage tests and cause them to break.

Another area that we are interested in investigating is related to IDE enhancements for updating tests and automatically reflecting developer changes applied to the web application. Such IDE enhancements could automatically update tests as the web

application evolves. Ultimately, this may help in the creation of novel record/replay tools that are more robust with respect to test breakages.

As noted earlier, taxonomies do typically evolve as additional observations of the world are made. While our initial attempts to validate our taxonomy revealed no shortcomings, it is likely that continued studies will reveal the need for adjustments.

Our taxonomy presents a quantitative and a qualitative assessment of the causes behind test breakages using Selenium IDE, which is a second generation testing tool. Second generation tools identify web elements to be selected within the user interface according to their position within the DOM tree and/or attributes. There are, however, two other generations of tools that have been used to test web applications. First generation tools rely on screen coordinates in order to locate and manipulate web elements within the web application under test, and have largely been deprecated due to the flakiness of coordinates. Third generation tools, (e.g., Sikuli) test web applications based on image recognition of web elements within the user interface of the web application under test. As future work, we plan to investigate test breakage taxonomies for third generation tools and other second generation tools (e.g., Selenium WebDriver).

Appendix A

Data Per Version:

Table A.1: Breakages Encountered for Each Pair of Versions of PHPFusion-Node 1.1.1 Not Expanded

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
V1-V2	1	0	3	1	0	0	0	0	0	0	0	1	0	0	6
V2-V3	0	0	2	3	0	0	0	0	0	0	0	0	0	0	5
V3-V4	1	0	0	0	0	1	0	0	0	0	0	0	0	0	2
V4-V5	2	0	0	1	0	0	0	0	0	0	0	1	0	1	5
V5-V6	2	0	0	1	0	0	0	0	0	0	0	0	0	0	3
V6-V7	1	0	0	1	0	0	0	1	0	1	0	0	0	0	4
V7-V8	0	1	0	1	0	0	0	0	0	0	0	0	0	0	2
V8-V9	0	0	0	1	1	1	0	0	0	0	0	0	0	0	3
V9-V10	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
V10-V11	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
V11-V12	2	0	0	0	0	0	1	0	0	0	1	0	0	0	4
V12-V13	0	0	0	1	0	1	0	0	0	0	0	0	0	0	2
V13-V14	2	0	0	2	0	0	0	0	0	0	0	1	0	0	5
V14-V15	1	1	0	1	0	0	0	0	0	0	0	0	0	0	3
V15-V16	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
V16-V17	0	1	1	1	0	0	0	0	0	0	0	0	0	0	3
V17-V18	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V18-V19	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V19-V20	2	0	0	1	0	0	0	0	0	1	0	0	0	0	4
V20-V21	2	0	2	1	0	0	2	0	0	0	0	0	0	0	7
V21-V22	1	1	1	0	0	0	0	0	0	0	0	0	0	0	3
V22-V23	1	0	0	1	0	0	0	0	0	0	0	0	0	1	3
V23-V24	1	0	0	1	0	0	0	0	0	0	0	0	0	0	2
V24-V25	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table A.2: Breakages Encountered for Each Pair of Versions of PHPFusion-Node 1.1.1 Not Expanded (Continued)

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
V25-V26	3	2	0	1	0	0	1	0	0	0	0	0	1	0	8
V26-V27	3	1	0	1	0	0	0	0	0	0	0	0	0	0	5
V27-V28	2	0	1	0	0	0	0	0	0	0	1	0	0	0	4
V28-V29	0	0	1	2	1	0	0	0	0	0	0	0	0	0	4
V29-V30	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
V30-V31	1	0	0	0	0	0	0	0	2	0	0	0	0	0	3
V31-V32	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
V32-V33	1	0	2	1	0	0	0	0	0	0	0	0	0	0	4
V33-V34	0	0	1	0	0	0	1	0	0	0	0	0	0	0	2
V34-V35	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
V35-V36	0	1	1	0	0	0	0	0	0	0	0	0	0	0	2
V36-V37	0	0	1	0	0	0	0	1	0	0	0	0	0	0	2
V37-V38	1	0	1	0	0	0	0	0	0	0	1	0	0	0	3
V38-V39	1	0	2	1	0	0	0	0	0	0	0	0	0	0	4
V39-V40	1	0	1	3	0	0	0	0	0	0	0	0	0	0	5
V40-V41	2	0	1	3	0	0	1	0	0	0	0	0	1	0	8
V41-V42	4	0	1	0	0	1	0	0	0	0	0	0	0	0	6
V42-V43	2	0	1	1	0	0	0	0	0	0	0	0	0	0	4
V43-V44	1	0	1	1	0	0	0	0	0	0	0	0	0	0	3
V44-V45	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
V45-V46	0	1	1	0	0	0	0	0	0	0	0	0	0	0	2
V46-V47	0	1	1	3	0	0	0	0	0	0	0	0	0	0	5
V47-V48	0	0	1	2	0	0	0	0	0	0	0	0	0	0	3
V48-V49	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
Total	45	10	33	39	2	4	6	2	2	2	3	3	2	2	155

Table A.3: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of PHPFusion

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
V1-V2	0	0	0	0	1	0	1	1	0	3
V2-V3	0	0	0	0	0	0	0	0	0	0
V3-V4	0	0	0	0	0	2	0	0	0	2
V4-V5	0	0	0	0	0	0	0	0	0	0
V5-V6	0	0	0	0	0	0	0	0	0	0
V6-V7	0	0	0	0	0	0	0	0	0	0
V7-V8	1	0	0	1	0	0	0	0	0	2
V8-V9	0	0	0	0	0	0	0	0	0	0
V9-V10	1	0	0	0	0	0	0	0	0	1
V10-V11	0	0	1	0	0	0	0	0	0	1
V11-V12	0	0	0	0	1	2	1	0	1	5
V12-V13	0	0	0	0	0	0	0	0	0	0
V13-V14	0	0	0	0	0	0	0	0	0	0
V14-V15	0	0	0	0	0	0	0	0	0	0
V15-V16	1	0	0	2	0	0	0	0	0	3
V16-V17	0	0	0	0	0	0	0	0	0	0
V17-V18	0	0	0	0	0	1	0	0	0	1
V18-V19	0	1	0	0	0	0	0	0	0	1
V19-V20	0	0	0	0	0	0	0	0	0	0
V20-V21	0	0	0	0	0	0	1	0	0	1
V21-V22	1	1	0	0	0	0	0	0	0	2
V22-V23	0	0	0	0	0	0	0	0	0	0
V23-V24	0	0	0	0	0	0	0	0	0	0
V24-V25	0	0	0	0	2	0	0	0	0	2

Table A.4: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of PHPFusion (Continued)

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
V25-V26	0	0	0	0	0	0	0	0	0	0
V26-V27	0	0	0	0	0	0	0	0	0	0
V27-V28	0	1	0	0	0	0	0	0	0	1
V28-V29	0	0	0	0	0	0	0	0	0	0
V29-V30	0	0	1	0	0	0	0	0	0	1
V30-V31	0	0	0	0	0	0	0	0	0	0
V31-V32	0	0	0	0	0	0	0	0	0	0
V32-V33	0	0	0	0	0	0	0	0	1	1
V33-V34	0	0	0	1	0	0	0	1	0	2
V34-V35	0	1	1	0	0	0	0	0	0	2
V35-V36	0	0	0	0	0	0	0	0	0	0
V36-V37	0	1	2	0	0	0	0	0	0	3
V37-V38	0	0	0	0	0	0	0	0	1	1
V38-V39	0	0	0	1	0	0	0	0	0	1
V39-V40	1	0	0	0	0	0	0	0	0	1
V40-V41	1	0	0	0	0	0	0	0	0	1
V41-V42	0	0	0	0	0	0	0	0	0	0
V42-V43	0	0	0	0	0	0	0	0	0	0
V43-V44	1	0	0	1	0	0	0	0	1	3
V44-V45	0	0	0	0	0	0	0	0	0	0
V45-V46	1	0	0	0	0	0	0	0	0	1
V46-V47	0	0	0	0	0	0	0	0	0	0
V47-V48	2	0	0	0	0	0	0	0	0	2
V48-V49	1	0	0	0	0	0	0	0	0	1
total	11	5	5	6	4	5	3	2	4	45

Table A.5: Breakages Encountered for Each Pair of Versions of PHPAddressBook-Node 1.1.1 Not Expanded

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
V1-V2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V2-V3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V3-V4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V4-V5	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
V5-V6	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
V6-V7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V7-V8	2	0	0	0	0	0	0	0	0	0	0	1	0	0	3
V8-V9	3	0	0	0	0	1	0	0	0	0	0	0	0	0	4
V9-V10	4	0	0	0	1	0	0	0	0	0	0	0	0	0	5
V10-V11	4	0	0	0	0	0	1	0	0	0	0	0	0	1	6
V11-V12	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V12-V13	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
V13-V14	1	0	2	0	0	0	0	0	0	1	0	0	0	0	4
V14-V15	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V15-V16	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V16-V17	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
V17-V18	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
V18-V19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V19-V20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V20-V21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V21-V22	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
V22-V23	0	0	0	0	0	1	0	0	0	0	0	0	1	0	2
V23-V24	5	0	0	0	0	0	0	0	0	0	0	0	0	0	5
V24-V25	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3
V25-V26	3	0	0	0	0	0	1	0	0	1	0	0	0	0	5
V26-V27	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3
V27-V28	0	0	2	0	0	0	0	0	0	0	0	0	0	0	2
V28-V29	1	1	0	0	0	0	0	0	0	0	0	0	0	0	2
V29-V30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V30-V31	1	0	0	0	0	1	0	0	0	0	0	0	0	0	2
V31-V32	0	0	1	0	0	0	1	0	0	0	0	0	0	0	2
V32-V33	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V33-V34	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V34-V35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V35-V36	2	1	1	0	0	0	0	0	0	0	0	0	1	0	5
V36-V37	3	0	0	1	0	0	0	0	0	0	0	0	0	0	4
V37-V38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V38-V39	3	0	1	0	0	0	1	0	0	0	0	0	0	0	5
V39-V40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.6: Breakages Encountered for Each Pair of Versions of PHPAddressBook-Node 1.1.1 Not Expanded (Continued)

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
V40-V41	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
V41-V42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V42-V43	0	1	0	0	0	1	1	0	0	0	0	0	0	0	3
V43-V44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V44-V45	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V45-V46	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
V46-V47	0	1	0	1	0	0	0	0	0	0	0	0	0	0	2
V47-V48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V48-V49	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
V49-V50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V50-V51	0	0	1	0	0	0	1	0	0	0	0	0	0	0	2
V51-V52	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
V52-V53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V53-V54	1	0	0	0	0	0	1	0	0	0	0	0	0	0	2
V54-V55	0	0	1	0	0	0	0	0	1	0	0	0	0	0	2
V55-V56	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
V56-V57	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V57-V58	0	0	1	1	0	0	0	0	0	0	0	0	0	0	2
V58-V59	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
V59-V60	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
V60-V61	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V61-V62	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V62-V63	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V63-V64	2	0	0	1	0	0	1	0	0	0	0	0	0	0	4
V64-V65	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V65-V66	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V66-V67	1	1	0	0	0	0	1	0	0	0	0	0	0	0	3
V67-V68	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
V68-V69	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
V69-V70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V70-V71	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
V71-V72	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
V72-V73	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
V73-V74	1	0	0	0	0	0	1	0	0	0	0	0	0	0	2
V74-V75	1	0	1	0	0	0	0	0	0	0	0	0	0	0	2
V75-V76	1	0	0	0	0	0	1	0	0	0	0	0	0	0	2
V76-V77	2	0	1	0	0	0	0	0	0	0	0	0	0	0	3
V77-V78	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V78-V79	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Total	63	7	16	6	3	5	16	1	1	2	1	1	2	1	125

Table A.7: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of PHPAddressBook

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
V1-V2	0	0	0	0	0	0	0	0	0	0
V2-V3	0	0	0	0	0	0	0	0	0	0
V3-V4	1	0	0	0	0	0	0	0	0	1
V4-V5	0	1	0	0	0	0	0	0	0	1
V5-V6	0	0	1	0	0	0	0	0	1	2
V6-V7	0	0	0	0	0	0	0	0	0	0
V7-V8	0	0	0	0	0	1	0	0	0	1
V8-V9	0	0	0	0	0	0	0	0	0	0
V9-V10	0	0	0	0	0	0	0	0	2	2
V10-V11	1	0	0	0	0	0	0	0	0	1
V11-V12	0	0	0	0	0	0	0	0	0	0
V12-V13	0	0	0	0	1	0	0	0	0	1
V13-V14	0	0	0	0	0	0	0	0	0	0
V14-V15	1	0	1	0	0	0	0	0	1	3
V15-V16	0	0	0	1	0	1	0	0	0	2
V16-V17	0	0	0	0	0	0	0	0	0	0
V17-V18	1	0	0	0	0	0	0	0	0	1
V18-V19	0	0	0	0	1	0	0	0	0	1
V19-V20	0	0	0	0	0	0	0	0	2	2
V20-V21	1	0	0	0	0	0	0	0	0	1
V21-V22	0	1	0	0	0	1	1	0	0	3
V22-V23	0	0	0	1	0	0	0	0	0	1
V23-V24	0	0	1	0	0	0	0	0	1	2
V24-V25	0	0	0	0	1	0	0	0	0	1
V25-V26	1	0	0	0	0	1	0	0	0	2
V26-V27	0	0	0	0	0	0	0	0	0	0
V27-V28	0	1	0	0	0	0	1	0	0	2
V28-V29	0	0	0	0	0	1	0	0	0	1
V29-V30	0	0	0	0	0	0	0	0	0	0
V30-V31	0	0	1	0	1	0	0	0	0	2
V31-V32	1	0	0	1	0	0	0	0	0	2
V32-V33	0	1	0	0	0	0	0	0	0	1
V33-V34	0	0	0	0	0	0	0	0	0	0
V34-V35	0	0	0	0	0	0	1	0	0	1
V35-V36	0	0	0	0	1	0	0	0	0	1
V36-V37	0	0	0	0	0	0	0	0	0	0
V37-V38	1	0	0	0	0	0	0	0	0	1
V38-V39	0	0	0	0	0	0	0	0	0	0
V39-V40	0	0	0	0	0	0	0	0	0	0

Table A.8: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of PHPAddressBook (Continued)

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
V40-V41	0	0	0	0	0	0	1	0	0	1
V41-V42	0	0	0	0	1	0	0	0	0	1
V42-V43	0	0	0	0	0	0	0	0	0	0
V43-V44	0	0	0	0	0	0	0	0	0	0
V44-V45	0	0	0	0	0	0	1	1	0	2
V45-V46	1	0	0	0	0	0	0	0	0	1
V46-V47	0	0	0	0	1	0	0	0	0	1
V47-V48	0	0	0	0	0	0	0	0	0	0
V48-V49	0	0	0	0	0	1	0	0	0	1
V49-V50	0	1	0	1	0	0	0	0	0	2
V50-V51	1	0	0	0	1	0	0	1	0	3
V51-V52	0	0	0	0	0	0	0	0	0	0
V52-V53	0	0	0	0	0	0	0	0	0	0
V53-V54	0	1	0	0	0	0	0	0	0	1
V54-V55	0	0	0	0	0	0	0	0	0	0
V55-V56	1	0	0	0	0	0	0	0	0	1
V56-V57	0	0	0	0	0	0	0	0	0	0
V57-V58	0	0	0	0	0	0	0	0	0	0
V58-V59	0	0	0	0	0	0	0	1	0	1
V59-V60	0	1	0	0	0	0	0	0	0	1
V60-V61	0	0	0	0	0	0	0	0	0	0
V61-V62	1	0	0	0	0	0	0	0	0	1
V62-V63	0	0	0	0	0	0	0	0	0	0
V63-V64	0	0	0	0	1	0	0	0	0	1
V64-V65	0	0	0	0	0	0	0	1	0	1
V65-V66	0	1	0	0	0	0	0	0	0	1
V66-V67	0	0	0	0	0	0	0	0	0	0
V67-V68	0	0	0	0	0	0	0	0	0	0
V68-V69	1	0	1	0	0	0	0	0	0	2
V69-V70	0	0	0	0	0	0	0	0	0	0
V70-V71	0	0	0	0	0	0	0	0	0	0
V71-V72	0	0	0	0	0	0	0	0	0	0
V72-V73	1	0	0	0	0	0	0	0	0	1
V73-V74	0	0	0	0	0	0	0	0	0	0
V74-V75	0	0	0	0	0	0	0	0	0	0
V75-V76	1	0	0	0	0	0	0	0	0	1
V76-V77	0	0	0	0	0	0	0	0	0	0
V77-V78	0	0	0	0	0	0	0	0	0	0
V78-V79	0	0	0	0	0	0	0	0	0	0
Total	15	8	5	4	9	6	5	4	7	63

Table A.9: Breakages Encountered for Each Pair of Versions of PHPAgenda-Node 1.1.1 Not Expanded

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
V1-V2	1	0	1	0	0	0	1	0	0	0	0	0	0	0	3
V2-V3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V3-V4	2	0	0	0	1	0	0	0	0	0	0	0	0	0	3
V4-V5	1	0	1	1	0	0	1	0	0	0	0	0	0	0	4
V5-V6	1	1	0	0	0	0	0	0	0	0	0	0	0	0	2
V6-V7	0	0	1	1	0	1	1	0	0	0	0	0	0	0	4
V7-V8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V8-V9	1	0	0	0	0	0	1	0	1	0	0	0	0	1	4
V9-V10	1	0	0	1	0	0	1	1	0	0	0	0	0	0	4
V10-V11	1	0	0	0	0	1	1	0	0	0	0	0	0	0	3
V11-V12	0	0	1	0	0	0	1	0	0	0	1	1	0	0	4
V12-V13	1	0	0	1	0	0	0	0	0	0	0	0	0	0	2
V13-V14	0	0	1	0	0	0	1	0	0	0	0	0	0	1	3
V14-V15	2	0	0	1	0	0	0	0	0	0	0	0	1	0	4
V15-V16	0	0	1	0	0	0	1	0	0	1	0	0	0	0	3
V16-V17	3	1	1	1	0	0	0	0	0	0	1	0	0	0	7
V17-V18	0	0	0	0	0	1	1	0	0	0	0	0	0	0	2
V18-V19	2	0	1	0	0	1	0	0	0	0	0	1	0	0	5
V19-V20	2	0	1	0	0	0	1	0	0	0	0	0	0	1	5
V20-V21	0	1	0	1	0	0	0	0	0	0	0	0	0	0	2
V21-V22	2	0	0	0	0	0	1	1	0	0	0	0	0	0	4
V22-V23	0	0	1	0	0	0	1	0	0	0	1	0	0	0	3
V23-V24	1	0	1	1	0	0	0	0	0	0	0	0	0	0	3
V24-V25	0	0	0	1	0	1	1	0	0	0	0	1	0	0	4
V25-V26	1	0	1	0	0	0	0	0	0	0	0	0	0	0	2
V26-V27	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
V27-V28	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V28-V29	1	0	1	1	0	1	1	0	0	0	0	0	0	0	5
V29-V30	1	0	0	0	0	0	1	0	0	0	0	0	0	0	2
V30-V31	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
V31-V32	3	0	1	0	0	0	1	0	0	0	0	0	0	0	5
Total	31	3	14	12	1	6	17	2	1	1	3	3	1	3	98

Table A.10: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of PHPAgenda

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
V1-V2	0	0	0	0	0	0	0	0	0	0
V2-V3	0	0	0	0	0	0	0	0	0	0
V3-V4	0	0	1	0	0	0	0	0	0	1
V4-V5	1	0	0	0	0	0	0	0	0	1
V5-V6	0	0	0	0	0	1	0	0	0	1
V6-V7	0	0	0	1	0	0	0	0	0	1
V7-V8	0	0	1	0	0	0	1	0	0	2
V8-V9	1	0	0	0	0	0	0	0	0	1
V9-V10	0	1	0	0	0	0	0	0	0	1
V10-V11	0	0	0	0	0	1	0	0	0	1
V11-V12	0	0	0	1	1	0	0	0	0	2
V12-V13	1	0	1	0	0	0	0	0	0	2
V13-V14	0	0	0	0	0	0	0	0	0	0
V14-V15	0	1	0	0	1	0	0	0	0	2
V15-V16	0	0	0	0	0	1	1	0	0	2
V16-V17	1	0	0	1	0	0	0	0	0	2
V17-V18	0	0	1	0	1	0	0	0	0	2
V18-V19	0	1	0	0	0	0	0	0	0	1
V19-V20	0	0	0	0	0	0	0	0	0	0
V20-V21	1	0	0	0	0	0	0	0	0	1
V21-V22	0	0	0	0	0	0	0	0	0	0
V22-V23	0	1	0	0	1	0	0	0	0	2
V23-V24	1	0	0	0	0	0	0	0	0	1
V24-V25	0	0	0	0	0	0	0	0	0	0
V25-V26	2	0	0	0	0	0	0	0	0	2
V26-V27	0	1	0	0	0	0	0	0	0	1
V27-V28	0	0	0	0	0	0	0	0	0	0
V28-V29	1	0	0	0	0	0	0	0	0	1
V29-V30	0	0	0	0	0	0	0	0	0	0
V30-V31	1	0	0	0	0	0	0	0	0	1
V31-V32	0	0	0	0	0	0	0	0	0	0
Total	10	5	4	3	4	3	2	0	0	31

Table A.11: Breakages Encountered for Each Pair of Versions of MyCollaboration-Node 1.1.1 Not Expanded

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
V1-V2	3	1	1	0	0	0	1	0	0	0	0	0	0	1	7
V2-V3	3	0	1	0	0	0	1	0	0	0	0	0	0	0	5
V3-V4	1	0	0	0	0	0	0	1	0	0	0	0	0	0	2
V4-V5	1	0	1	1	1	0	1	0	0	0	0	0	0	0	5
V5-V6	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
V6-V7	2	2	1	1	0	1	1	0	1	0	0	0	1	0	10
V7-V8	2	0	0	1	0	0	0	0	0	0	0	0	0	0	3
V8-V9	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3
V9-V10	1	0	1	1	1	0	1	0	0	1	0	0	0	0	6
V10-V11	3	2	0	0	0	0	0	1	0	0	0	0	0	0	6
V11-V12	1	0	0	1	0	0	1	0	0	0	0	1	0	1	5
V12-V13	3	2	0	1	0	1	0	0	0	0	0	0	1	0	8
V13-V14	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V14-V15	1	0	1	0	0	0	1	0	0	0	1	0	0	0	4
V15-V16	3	0	1	0	0	1	1	0	0	0	0	0	0	1	7
V16-V17	2	1	1	1	0	0	0	0	0	0	0	0	0	0	5
V17-V18	3	0	1	0	0	0	0	0	0	0	0	0	1	0	5
V18-V19	3	0	1	1	0	0	0	0	0	0	0	0	0	0	5
V19-V20	2	0	1	0	0	0	1	0	0	0	0	0	0	0	4
V20-V21	0	0	1	0	0	0	0	0	0	0	0	0	0	1	2
V21-V22	3	0	1	0	0	0	0	0	0	0	0	0	0	0	4
V22-V23	1	0	1	1	0	0	0	0	0	0	0	0	0	0	3
V23-V24	2	0	1	1	0	0	0	0	0	0	0	0	0	0	4
V24-V25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V25-V26	3	0	1	1	0	0	0	0	0	0	0	0	0	1	6
V26-V27	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V27-V28	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V28-V29	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Total	53	8	16	11	2	3	9	2	1	1	1	1	3	6	117

Table A.12: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of MyCollaboration

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
V1-V2	0	0	0	0	0	0	0	0	0	0
V2-V3	1	0	0	0	0	0	0	1	0	2
V3-V4	0	0	1	0	0	0	1	0	0	2
V4-V5	1	0	0	0	1	0	0	0	1	3
V5-V6	2	0	0	0	0	0	0	0	1	3
V6-V7	0	0	0	1	0	1	0	0	0	2
V7-V8	3	1	1	0	1	0	1	1	0	8
V8-V9	1	0	0	0	0	0	0	0	0	1
V9-V10	0	0	0	1	0	0	0	0	1	2
V10-V11	0	1	0	0	0	0	1	0	0	2
V11-V12	0	0	1	0	1	0	0	0	1	3
V12-V13	0	0	0	0	0	0	0	1	0	1
V13-V14	0	0	0	1	0	1	0	0	0	2
V14-V15	0	1	0	0	1	0	0	0	1	3
V15-V16	1	0	1	0	0	0	0	1	0	3
V16-V17	0	0	0	0	0	0	0	0	0	0
V17-V18	0	0	0	0	0	0	0	0	1	1
V18-V19	0	1	0	0	1	0	1	0	0	3
V19-V20	0	0	1	0	0	0	0	1	0	2
V20-V21	1	0	1	0	0	0	0	0	0	2
V21-V22	0	1	0	0	0	0	0	0	1	2
V22-V23	0	0	0	0	0	1	0	1	0	2
V23-V24	0	0	1	0	0	0	0	0	1	2
V24-V25	0	0	0	0	0	0	0	1	0	1
V25-V26	0	0	0	0	0	0	0	0	0	0
V26-V27	0	0	0	0	0	0	0	0	1	1
V27-V28	0	0	0	0	0	0	0	0	0	0
V28-V29	0	0	0	0	0	0	0	0	0	0
Total	10	5	7	3	5	3	4	7	9	53

Table A.13: Breakages Encountered for Each Pair of Versions of Joomla-Node 1.1.1 Not Expanded

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
V1-V2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
V2-V3	0	0	0	1	0	1	0	0	0	1	0	0	0	0	3
V3-V4	1	0	1	0	0	0	0	0	0	0	0	0	1	0	3
V4-V5	0	0	0	0	0	0	1	0	0	0	0	0	0	1	2
V5-V6	0	3	0	1	0	0	0	0	1	0	0	0	0	0	5
V6-V7	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
V7-V8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V8-V9	0	0	1	0	0	0	0	0	0	0	0	1	0	0	2
V9-V10	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
V10-V11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V11-V12	1	0	0	0	0	1	0	0	0	0	0	0	0	0	2
V12-V13	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
V13-V14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V14-V15	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V15-V16	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
V16-V17	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
V17-V18	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V18-V19	1	0	0	1	0	0	0	0	0	0	0	0	0	0	2
V19-V20	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
V20-V21	2	0	0	0	0	0	0	0	0	1	0	0	0	0	3
V21-V22	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
V22-V23	3	3	0	0	0	1	0	0	0	0	0	0	0	0	7
V23-V24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V24-V25	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
V25-V26	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3
V26-V27	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3
V27-V28	2	0	0	0	0	1	0	0	0	0	0	0	0	0	3
V28-V29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V29-V30	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V30-V31	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V31-V32	3	2	0	0	0	0	0	0	0	0	0	0	1	0	6
V32-V33	2	0	0	1	0	0	0	0	0	0	0	0	0	0	3
V33-V34	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V34-V35	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V35-V36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V36-V37	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V37-V38	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1

Table A.14: Breakages Encountered for Each Pair of Versions of Joomla-Node 1.1.1 Not Expanded (Continued)

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
V38-V39	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
V39-V40	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
V40-V41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V41-V42	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
V42-V43	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3
V43-V44	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
V44-V45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V45-V46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V46-V47	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
V47-V48	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V48-V49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V49-V50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V50-V51	0	1	0	1	0	0	0	0	0	0	0	0	0	0	2
V51-V52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V52-V53	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V53-V54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V54-V55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V55-V56	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3
V56-V57	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V57-V58	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V58-V59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V59-V60	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V60-V61	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
V61-V62	1	0	0	1	0	0	0	0	0	0	0	0	0	0	2
V62-V63	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
V63-V64	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V64-V65	0	0	1	1	0	0	0	0	0	0	0	0	0	0	2
V65-V66	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V66-V67	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
V67-V68	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V68-V69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V69-V70	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
V70-V71	3	0	0	0	0	1	0	0	0	0	0	0	0	0	4
V71-V72	3	0	1	0	0	0	0	0	0	0	0	0	0	0	4
V72-V73	2	0	0	0	0	1	0	0	0	0	0	0	0	0	3
V73-V74	4	0	0	0	0	0	0	0	0	0	0	0	0	0	4

Table A.15: Breakages Encountered for Each Pair of Versions of Joomla-Node 1.1.1 Not Expanded (Continued)

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
V74-V75	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
V75-V76	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V76-V77	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V77-V78	4	0	0	0	0	1	0	0	0	0	0	0	0	0	5
V78-V79	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
V70-V80	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
V80-V81	0	1	0	0	0	1	0	0	0	0	0	0	1	0	3
V81-V82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V82-V83	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
V83-V84	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3
V84-V85	0	0	0	0	0	1	0	0	0	0	0	0	1	0	2
V85-V86	1	0	0	0	0	0	1	0	0	0	0	0	0	0	2
V86-V87	4	0	0	0	0	0	0	0	0	1	0	0	0	0	5
V87-V88	3	1	0	0	0	0	0	0	0	0	0	0	0	0	4
V88-V89	1	0	0	0	0	0	0	0	0	0	0	0	1	0	2
V89-V90	3	0	0	0	0	0	0	0	0	0	0	0	0	1	4
V90-V91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V91-V92	3	0	0	0	0	1	0	0	0	0	0	0	0	0	4
V92-V93	4	1	0	0	0	0	0	0	0	0	0	0	0	0	5
V93-V94	5	0	0	0	0	0	0	0	0	0	0	0	0	0	5
V94-V95	3	0	0	0	0	1	0	1	0	0	0	0	0	0	5
V95-V96	4	0	0	0	0	0	0	0	0	0	0	0	1	0	5
V96-V97	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V97-V98	4	0	1	0	0	0	0	0	0	0	0	0	0	0	5
V98-V99	3	0	0	0	0	1	1	0	0	0	0	0	0	0	5
V99-V100	3	0	1	0	0	0	0	0	0	0	0	0	1	0	5
V100-V101	3	0	0	0	0	0	0	0	0	1	0	0	0	0	4
V101-V102	4	0	0	0	0	0	0	0	0	0	0	0	0	0	4
V102-V103	1	1	0	0	0	0	0	0	0	0	0	0	0	1	3
V103-V104	4	0	0	0	0	1	0	0	0	0	0	0	0	0	5
V104-V105	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V105-V106	3	0	1	0	0	0	0	0	0	0	0	0	0	0	4
V106-V107	2	0	0	0	0	1	0	0	1	0	0	0	0	0	4
V107-V108	4	0	1	1	0	0	0	0	0	0	0	0	0	0	6
V108-V109	3	0	0	0	0	1	0	0	0	0	0	0	1	0	5
V109-V110	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V110-V111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Total	138	15	11	11	1	18	4	3	2	6	1	1	13	3	227

Table A.16: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of Joomla

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
V1-V2	0	0	0	0	0	0	0	0	1	1
V2-V3	0	0	0	0	0	0	0	0	0	0
V3-V4	1	0	0	1	0	0	0	0	0	2
V4-V5	0	1	0	0	0	0	1	0	1	3
V5-V6	0	0	0	1	0	0	0	1	0	2
V6-V7	0	0	1	0	0	0	0	0	0	1
V7-V8	0	0	0	1	0	1	0	0	0	2
V8-V9	1	0	0	0	2	0	0	0	0	3
V9-V10	0	0	0	1	0	0	0	0	0	1
V10-V11	0	1	0	0	0	0	1	0	0	2
V11-V12	0	0	0	0	0	0	0	0	0	0
V12-V13	1	0	0	0	0	0	0	0	0	1
V13-V14	0	0	1	0	0	0	0	0	1	2
V14-V15	0	0	0	1	0	0	0	0	0	1
V15-V16	0	1	0	1	0	0	0	0	0	2
V16-V17	1	0	0	0	0	1	0	0	0	2
V17-V18	0	0	0	0	0	0	0	0	0	0
V18-V19	0	0	0	0	0	0	0	0	0	0
V19-V20	0	0	0	1	0	0	0	1	0	2
V20-V21	1	0	1	0	0	0	0	0	0	2
V21-V22	0	0	0	0	0	0	0	0	0	0
V22-V23	0	1	0	0	0	1	0	0	0	2
V23-V24	1	0	0	1	2	0	0	0	0	4
V24-V25	0	0	1	0	0	0	0	0	0	1
V25-V26	0	0	0	0	0	0	0	0	0	0
V26-V27	0	0	1	1	0	0	0	0	0	2
V27-V28	1	0	0	0	0	0	0	0	0	1
V28-V29	0	0	0	0	0	0	1	0	0	1
V29-V30	0	1	0	0	0	0	0	0	1	2
V30-V31	0	0	0	1	0	0	0	0	0	1
V31-V32	0	0	0	0	0	0	0	0	0	0
V32-V33	1	0	0	0	0	0	0	0	0	1
V33-V34	0	0	0	0	0	0	0	0	0	0
V34-V35	0	0	0	0	0	0	0	0	1	1
V35-V36	0	0	0	1	0	0	0	0	0	1
V36-V37	0	0	0	0	0	0	0	0	0	0
V37-V38	0	0	0	0	0	0	0	0	0	0

Table A.17: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of Joomla (Continued)

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
V38-V39	0	1	0	0	2	0	0	0	0	3
V39-V40	0	0	0	1	0	0	0	0	0	1
V40-V41	0	0	0	0	0	0	1	1	0	2
V41-V42	0	0	1	0	0	0	0	0	0	1
V42-V43	0	0	0	0	0	0	0	0	0	0
V43-V44	0	0	0	0	0	0	0	0	0	0
V44-V45	1	0	0	0	0	0	1	0	1	3
V45-V46	0	0	0	0	0	0	0	0	0	0
V46-V47	0	0	0	0	0	0	0	0	0	0
V47-V48	0	0	0	0	0	0	0	1	0	1
V48-V49	0	0	0	0	0	0	0	0	0	0
V49-V50	0	0	0	0	0	0	0	0	0	0
V50-V51	0	0	1	0	0	0	0	0	0	1
V51-V52	1	0	0	0	0	0	0	0	1	2
V52-V53	0	0	0	0	0	0	0	0	0	0
V53-V54	0	0	0	1	0	0	0	0	0	1
V54-V55	0	0	0	0	0	0	0	0	0	0
V55-V56	0	0	0	1	0	0	0	0	0	1
V56-V57	0	0	0	0	0	0	0	0	0	0
V57-V58	1	0	0	0	0	0	0	0	0	1
V58-V59	0	0	0	1	0	0	0	0	0	1
V59-V60	0	0	0	0	0	0	0	0	0	0
V60-V61	0	0	0	0	0	0	0	0	0	0
V61-V62	0	0	0	0	0	0	0	0	0	0
V62-V63	0	0	0	1	0	0	0	0	0	1
V63-V64	0	0	0	1	0	0	0	0	0	1
V64-V65	0	0	0	0	0	0	1	0	0	1
V65-V66	1	0	0	0	0	0	0	0	0	1
V66-V67	0	0	0	1	0	0	0	1	0	2
V67-V68	0	0	0	0	0	0	0	0	0	0
V68-V69	0	1	1	0	0	0	0	0	0	2
V69-V70	0	0	0	0	0	0	0	0	0	0
V70-V71	0	1	0	1	0	0	0	0	0	2
V71-V72	0	0	0	0	1	0	0	0	0	1
V72-V73	0	0	1	0	0	0	1	1	1	4
V73-V74	0	0	0	0	0	1	0	0	0	1

Table A.18: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of Joomla (Continued)

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
V74-V75	1	0	0	0	0	0	0	0	0	1
V75-V76	0	0	0	0	0	0	0	0	0	0
V76-V77	0	0	1	0	0	0	0	0	0	1
V77-V78	1	1	0	0	0	0	0	0	0	2
V78-V79	0	0	0	0	0	0	0	1	0	1
V70-V80	0	0	1	0	0	0	0	0	1	2
V80-V81	0	0	0	0	1	0	0	1	0	2
V81-V82	0	1	0	0	0	0	0	0	0	1
V82-V83	0	0	0	0	0	0	0	0	0	0
V83-V84	0	0	0	0	0	0	0	1	1	2
V84-V85	0	0	0	0	0	0	1	0	0	1
V85-V86	1	1	1	1	0	0	0	0	0	4
V86-V87	0	0	0	0	0	0	0	0	0	0
V87-V88	0	0	0	0	1	0	0	0	0	1
V88-V89	1	0	0	0	0	0	0	1	0	2
V89-V90	0	0	0	0	0	0	1	0	0	1
V90-V91	0	0	0	0	0	1	0	0	0	1
V91-V92	1	0	0	0	0	0	0	0	0	1
V92-V93	0	0	0	0	0	0	0	0	1	1
V93-V94	0	0	1	0	0	0	0	1	0	2
V94-V95	0	0	0	0	0	0	0	0	0	0
V95-V96	0	1	1	1	0	0	0	0	1	4
V96-V97	0	0	0	0	0	0	0	0	0	0
V97-V98	0	0	1	0	0	0	0	0	0	1
V98-V99	0	0	1	0	1	1	0	0	0	3
V99-V100	1	0	0	0	0	0	0	0	0	1
V100-V101	0	0	1	1	0	0	0	0	1	3
V101-V102	0	0	0	0	0	0	1	0	0	1
V102-V103	0	1	1	0	0	0	0	0	0	2
V103-V104	0	0	0	0	0	1	0	1	0	2
V104-V105	1	0	0	0	0	0	0	0	0	1
V105-V106	0	1	0	0	2	0	0	0	0	3
V106-V107	1	0	0	0	0	0	0	0	1	2
V107-V108	0	1	1	0	0	1	0	0	0	3
V108-V109	1	0	0	0	1	0	0	1	0	3
V109-V110	0	0	0	0	0	0	1	0	1	2
V110-V111	0	0	0	0	0	0	0	1	0	1
Total	21	15	19	22	13	8	11	14	15	138

Table A.19: Breakages Encountered for Each Pair of Versions of MyMovieLibrary-Node 1.1.1 Not Expanded

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
C1-C2	1	0	1	0	0	0	0	0	0	0	0	1	0	0	3
C2-C3	2	0	1	1	0	0	0	0	0	0	0	0	0	1	5
C3-C4	2	2	0	1	0	0	0	0	0	0	0	0	0	0	5
C4-C5	3	0	1	0	1	0	0	0	0	0	0	0	0	0	5
C5-C6	1	0	0	0	0	0	0	0	0	1	0	0	1	0	3
C6-C7	0	0	0	0	0	1	1	1	1	0	0	0	0	0	4
C7-C8	2	0	1	0	0	0	0	0	0	0	0	0	0	0	3
C8-C9	3	0	1	0	0	0	0	0	0	0	1	0	0	1	6
C9-C10	2	2	0	2	0	1	0	0	0	1	0	0	0	0	8
C10-C11	2	0	0	1	1	0	1	0	0	0	0	0	0	0	5
C11-C12	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
C12-C13	0	0	0	2	0	0	1	1	0	0	0	0	0	0	4
C13-C14	2	0	0	0	0	1	0	0	0	0	0	0	1	1	5
C14-C15	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3
C15-C16	1	0	1	2	0	1	1	0	0	0	0	0	0	0	6
C16-C17	4	0	0	0	0	0	0	0	0	0	0	0	0	0	4
C17-C18	2	0	1	0	0	0	0	0	0	0	0	0	0	1	4
C18-C19	2	0	0	0	0	0	1	0	0	0	0	0	0	0	3
C19-C20	2	0	2	3	0	0	0	0	0	0	0	0	0	0	7
C20-C21	1	0	0	0	0	0	1	1	0	0	0	0	0	0	3
C21-C22	1	0	3	1	0	1	0	0	0	0	0	0	0	0	6
C22-C23	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
Total	38	4	12	13	2	5	6	3	2	2	1	1	2	4	95

Table A.20: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of MyMovieLibrary

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
C1-C2	0	0	0	0	0	0	0	0	0	0
C2-C3	0	1	0	0	0	1	0	0	0	2
C3-C4	0	0	0	0	1	0	0	1	0	2
C4-C5	0	0	0	1	0	0	0	0	0	1
C5-C6	0	0	0	0	0	0	0	0	1	1
C6-C7	0	0	0	0	0	0	1	0	0	1
C7-C8	1	1	1	1	0	0	0	1	0	5
C8-C9	0	0	0	0	1	0	0	0	1	2
C9-C10	0	0	0	0	0	0	0	0	0	0
C10-C11	1	0	1	0	0	0	0	1	1	4
C11-C12	0	0	0	1	0	0	1	0	0	2
C12-C13	0	1	0	0	1	0	0	0	0	2
C13-C14	1	0	1	0	0	0	0	0	1	3
C14-C15	0	0	0	0	0	0	0	2	0	2
C15-C16	0	1	1	1	0	0	0	0	0	3
C16-C17	2	0	0	0	0	1	0	0	0	3
C17-C18	0	1	0	0	0	0	0	1	0	2
C18-C19	0	1	1	0	0	0	0	0	0	2
C19-C20	0	0	0	0	0	0	0	1	0	1
C20-C21	0	0	0	0	0	0	0	0	0	0
C21-C22	0	0	0	0	0	0	0	0	0	0
C22-C23	0	0	0	0	0	0	0	0	0	0
Total	5	6	5	4	3	2	2	7	4	38

Table A.21: Breakages Encountered for Each Pair of Versions of Dolibarr-Node 1.1.1 Not Expanded

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
C1-C2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
C2-C3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
C3-C4	2	0	1	1	0	0	0	0	0	0	0	0	0	0	4
C4-C5	2	1	0	0	0	0	1	0	1	0	0	0	0	0	5
C5-C6	3	0	0	1	0	0	0	0	0	0	0	0	0	0	4
C6-C7	3	2	1	0	0	1	0	1	0	0	0	0	0	0	8
C7-C8	1	0	0	0	0	1	0	0	0	0	1	0	0	0	3
C8-C9	2	0	0	1	0	0	1	0	1	0	0	0	1	1	7
C9-C10	3	1	1	0	0	1	0	0	0	0	0	0	0	0	6
C10-C11	3	0	0	0	0	0	0	2	0	0	0	0	0	0	5
C11-C12	1	2	0	0	0	0	0	0	0	1	0	0	0	0	4
C12-C13	1	0	1	0	0	0	0	0	1	0	0	0	0	0	3
C13-C14	1	0	0	1	0	0	1	0	0	0	0	0	0	0	3
C14-C15	3	1	1	0	0	0	0	0	0	0	0	0	0	0	5
C15-C16	1	0	0	0	0	1	0	0	1	0	0	0	0	0	3
C16-C17	1	0	1	0	0	0	0	0	0	1	1	0	0	0	4
C17-C18	2	1	0	0	0	0	1	0	0	0	0	0	0	0	4
C18-C19	2	0	0	1	0	0	0	0	0	0	0	1	0	0	4
C19-C20	1	1	1	0	0	0	0	0	0	0	0	0	0	0	3
C20-C21	2	0	1	0	0	0	1	0	0	0	0	0	0	0	4
C21-C22	2	0	0	1	0	0	0	0	0	0	0	0	0	0	3
C22-C23	1	0	1	0	0	0	1	0	0	0	1	0	0	0	4
C23-C24	2	0	0	0	1	0	1	0	0	0	0	0	0	0	4
C24-C25	2	0	1	1	0	0	1	0	0	0	0	0	0	0	5
C25-C26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C26-C27	1	0	1	0	0	0	0	0	0	0	0	0	0	0	2
C27-C28	2	0	1	0	0	0	0	0	0	0	0	0	0	0	3
C28-C29	2	0	1	0	0	0	0	0	0	0	0	0	0	0	3
C29-C30	2	0	1	0	0	0	0	0	0	0	0	0	0	0	3
Total	52	9	14	7	1	4	8	3	4	2	3	1	1	1	110

Table A.22: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of Dolibarr

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
C1-C2	0	0	0	0	0	0	0	0	0	0
C2-C3	0	1	1	1	0	0	0	0	0	3
C3-C4	0	0	0	0	0	1	0	0	0	1
C4-C5	0	0	1	0	1	0	1	1	0	4
C5-C6	0	1	0	0	0	0	0	0	0	1
C6-C7	1	0	0	0	0	0	0	0	0	1
C7-C8	0	1	0	0	0	0	0	1	0	2
C8-C9	1	0	0	0	0	0	0	0	0	1
C9-C10	0	0	1	0	0	1	0	0	1	3
C10-C11	1	0	0	0	0	1	0	0	0	2
C11-C12	0	0	0	0	1	0	1	2	0	4
C12-C13	0	0	0	0	0	1	0	0	0	1
C13-C14	0	0	0	0	0	0	0	0	1	1
C14-C15	0	1	0	0	0	0	1	0	0	2
C15-C16	0	0	0	1	1	0	0	0	1	3
C16-C17	0	0	1	0	0	0	0	0	0	1
C17-C18	1	0	0	0	0	1	0	0	0	2
C18-C19	0	0	0	1	1	0	1	1	0	4
C19-C20	0	1	0	0	1	0	0	0	0	2
C20-C21	0	0	1	0	1	0	1	0	0	3
C21-C22	0	0	0	0	0	0	0	1	1	2
C22-C23	0	0	1	0	0	1	1	0	0	3
C23-C24	0	1	0	0	1	0	0	0	0	2
C24-C25	1	0	0	1	0	0	2	0	0	4
C25-C26	0	0	0	0	0	0	0	0	0	0
C26-C27	0	0	0	0	0	0	0	0	0	0
C27-C28	0	0	0	0	0	0	0	0	0	0
C28-C29	0	0	0	0	0	0	0	0	0	0
C29-C30	0	0	0	0	0	0	0	0	0	0
Total	5	6	6	4	7	6	8	6	4	52

Table A.23: Breakages Encountered for Each Pair of Versions of YourContacts-Node 1.1.1 Not Expanded

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
C1-C2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
C2-C3	2	0	1	0	0	0	0	0	0	0	0	0	0	0	3
C3-C4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
C4-C5	0	1	0	0	0	0	0	1	1	0	0	0	1	1	5
C5-C6	1	0	0	0	1	1	0	0	0	0	0	1	0	0	4
C6-C7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C7-C8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C8-C9	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3
C9-C10	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
C10-C11	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
C11-C12	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
C12-C13	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
C13-C14	1	1	0	0	0	0	0	0	0	0	0	0	0	0	2
C14-C15	0	0	0	0	0	1	0	0	0	0	0	1	0	0	2
C15-C16	0	0	1	1	0	0	0	0	0	0	0	0	0	0	2
C16-C17	1	0	0	0	0	0	1	1	0	0	0	0	0	0	3
C17-C18	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
C18-C19	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
C19-C20	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
C20-C21	2	0	1	0	0	0	0	0	0	0	0	0	0	0	3
C21-C22	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3
C22-C23	1	1	0	1	0	1	0	0	0	1	0	0	0	0	5
C23-C24	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
C24-C25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C25-C26	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
C26-C27	1	0	0	0	0	0	1	0	0	0	0	0	0	0	2
C27-C28	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
C28-C29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C29-C30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C30-C31	0	0	0	1	0	0	0	1	0	0	0	0	0	0	2
C31-C32	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
C32-C33	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
C33-C34	1	1	0	0	0	0	0	0	0	0	0	0	0	1	3

Table A.24: Breakages Encountered for Each Pair of Versions of YourContacts-Node 1.1.1 Not Expanded (Continued)

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
C34-C35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C35-C36	1	0	0	0	1	0	1	0	0	0	0	0	0	0	3
C36-C37	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
C37-C38	1	0	0	1	0	0	0	0	0	0	0	0	0	0	2
C38-C39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C39-C40	0	1	1	0	0	1	0	0	0	0	0	0	0	0	3
C40-C41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C41-C42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C42-C43	1	0	0	0	0	0	0	0	1	0	0	0	0	0	2
C43-C44	0	0	1	0	0	0	1	0	0	0	0	0	0	0	2
C44-C45	0	0	0	1	0	0	0	0	0	1	0	0	0	0	2
C45-C46	0	1	0	0	0	1	0	0	0	0	0	0	0	0	2
C46-C47	2	0	0	0	0	0	0	0	0	0	0	0	1	0	3
C47-C48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C48-C49	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
C49-C50	0	0	1	0	0	1	0	0	1	0	0	0	0	0	3
C50-C51	1	0	0	0	0	0	0	0	0	0	0	0	1	0	2
C51-C52	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
C52-C53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C53-C54	1	0	0	0	0	0	1	0	0	0	0	0	0	0	2
C54-C55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C55-C56	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
C56-C57	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3
C57-C58	0	0	1	1	0	0	0	0	0	0	0	0	0	0	2
C58-C59	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
C59-C60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C60-C61	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
C61-C62	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
C62-C63	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
C63-C64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C64-C65	1	0	0	0	0	0	0	0	0	0	1	0	0	0	2
C65-C66	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.25: Breakages Encountered for Each Pair of Versions of YourContacts-Node 1.1.1 Not Expanded (Continued)

	1.1.1	1.1.2	1.1.3	1.1.4	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	5.1	5.2	Total
C66-C67	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C67-C68	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
C68-C69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C69-C70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C70-C71	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C71-C72	1	0	0	0	0	0	0	0	0	1	0	0	0	0	2
C72-C73	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
C73-C74	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
C74-C75	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
C75-C76	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
C76-C77	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
C77-C78	1	0	0	1	0	0	0	0	0	0	0	0	0	0	2
C78-C79	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C70-C80	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
C80-C81	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
C81-C82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C82-C83	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C83-C84	0	0	1	1	0	0	0	0	0	0	0	0	0	0	2
C84-C85	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C85-C86	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
C86-C87	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
C87-C88	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
C88-C89	0	0	1	0	0	1	0	0	0	0	0	0	0	0	2
C89-C90	3	0	1	0	0	0	0	0	0	0	0	0	0	0	4
C90-C91	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
C91-C92	1	0	0	1	0	0	0	0	0	0	0	0	0	0	2
C92-C93	1	0	0	0	0	1	0	0	0	0	0	0	0	0	2
C93-C94	1	0	1	0	0	0	0	0	0	0	0	0	0	0	2
C94-C95	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
C95-C96	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
C96-C97	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
C97-C98	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
C98-C99	1	0	1	1	0	0	0	0	0	0	0	0	0	0	3
C99-C100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Total	54	7	15	17	3	9	7	4	4	5	2	2	5	4	138

Table A.26: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of YourContacts

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
C1-C2	0	0	0	0	0	0	0	0	0	0
C2-C3	0	0	0	0	0	0	0	0	0	0
C3-C4	0	0	0	0	0	0	0	0	0	0
C4-C5	0	0	0	0	0	1	0	0	0	1
C5-C6	0	0	0	1	0	0	1	0	1	3
C6-C7	0	0	0	0	0	0	0	1	0	1
C7-C8	0	0	0	0	1	0	0	0	0	1
C8-C9	0	0	0	0	0	0	0	0	0	0
C9-C10	0	0	0	0	0	0	0	0	0	0
C10-C11	0	0	0	0	0	0	0	0	0	0
C11-C12	0	0	0	0	0	0	0	0	0	0
C12-C13	1	0	0	1	0	0	0	0	0	2
C13-C14	0	1	0	0	0	0	0	0	0	1
C14-C15	0	0	0	0	0	0	0	0	0	0
C15-C16	0	0	0	0	0	0	0	0	0	0
C16-C17	0	0	0	0	0	0	0	0	0	0
C17-C18	0	0	0	0	0	0	0	0	0	0
C18-C19	1	0	0	0	0	0	1	0	0	2
C19-C20	0	0	0	0	0	1	0	0	0	1
C20-C21	0	0	1	1	0	0	0	0	0	2
C21-C22	0	0	0	0	0	0	0	0	0	0
C22-C23	0	0	0	0	0	0	0	0	0	0
C23-C24	0	0	0	0	0	0	0	0	0	0
C24-C25	0	1	0	0	0	0	0	0	0	1
C25-C26	0	0	0	0	0	0	0	0	0	0
C26-C27	0	0	0	0	0	0	0	0	0	0
C27-C28	0	0	0	1	0	0	0	0	0	1
C28-C29	1	0	0	0	0	0	0	0	0	1
C29-C30	0	0	0	0	0	0	0	0	0	0
C30-C31	0	0	0	0	0	0	1	0	1	2
C31-C32	0	0	0	0	0	0	0	0	0	0
C32-C33	0	0	1	0	1	0	0	0	0	2
C33-C34	0	0	0	0	0	0	0	0	0	0
C34-C35	0	0	0	0	0	0	0	0	0	0

Table A.27: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of YourContacts (Continued)

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
C35-C36	0	0	0	0	0	0	0	0	0	0
C36-C37	0	1	0	0	0	0	0	0	0	1
C37-C38	0	0	1	0	0	0	0	0	0	1
C38-C39	0	0	0	1	0	0	0	0	0	1
C39-C40	0	0	0	0	0	0	0	0	0	0
C40-C41	0	0	0	0	0	0	0	0	0	0
C41-C42	0	0	0	0	0	0	0	0	0	0
C42-C43	0	0	0	0	0	0	1	0	0	1
C43-C44	0	0	0	0	0	0	0	0	0	0
C44-C45	0	0	0	0	0	0	0	0	0	0
C45-C46	0	0	0	0	0	0	0	0	0	0
C46-C47	0	0	0	0	0	0	0	0	0	0
C47-C48	0	0	0	0	0	0	0	0	0	0
C48-C49	0	0	0	0	0	0	0	0	0	0
C49-C50	0	0	0	0	1	0	0	0	0	1
C50-C51	0	0	0	0	0	0	0	0	0	0
C51-C52	0	0	0	0	0	0	0	0	0	0
C52-C53	0	0	0	0	0	0	0	0	0	0
C53-C54	0	0	0	0	0	0	0	0	0	0
C54-C55	0	0	0	0	0	0	0	0	0	0
C55-C56	0	0	0	0	0	0	0	0	1	1
C56-C57	0	0	0	0	0	0	1	0	0	1
C57-C58	1	0	0	0	0	0	0	0	0	1
C58-C59	0	0	0	0	0	1	0	0	0	1
C59-C60	0	0	0	0	0	0	0	0	0	0
C60-C61	0	0	0	0	0	0	1	0	0	1
C61-C62	0	0	0	0	0	0	0	0	0	0
C62-C63	0	0	0	0	0	0	0	0	0	0
C63-C64	0	0	0	0	1	0	0	1	0	2
C64-C65	0	0	0	0	0	0	0	0	0	0
C65-C66	0	0	0	1	0	0	0	0	0	1
C66-C67	0	0	0	0	0	0	0	0	0	0

Table A.28: "Element Attribute not Found" Breakages Encountered for Each Pair of Versions of YourContacts (Continued)

	1.1.1.1	1.1.1.2	1.1.1.3	1.1.1.4	1.1.1.5	1.1.1.6	1.1.1.7	1.1.1.8	1.1.1.9	Total
C67-C68	1	0	1	0	0	0	0	0	0	2
C68-C69	0	0	0	0	0	0	0	0	0	0
C69-C70	0	0	0	0	0	0	0	0	0	0
C70-C71	0	0	0	0	0	0	0	0	0	0
C71-C72	0	0	0	0	0	0	0	0	0	0
C72-C73	0	1	0	0	0	0	0	0	0	1
C73-C74	0	0	0	0	0	0	0	0	0	0
C74-C75	0	0	0	0	0	0	0	0	0	0
C75-C76	0	0	0	0	0	0	0	0	0	0
C76-C77	0	0	0	0	0	0	0	0	0	0
C77-C78	0	0	0	0	0	0	0	0	0	0
C78-C79	0	0	0	0	0	0	1	0	0	1
C70-C80	0	0	0	0	0	0	0	0	0	0
C80-C81	1	0	0	0	0	0	0	0	0	1
C81-C82	0	0	0	0	0	0	0	0	0	0
C82-C83	0	0	0	0	0	0	1	0	0	1
C83-C84	0	0	0	0	0	0	0	0	0	0
C84-C85	0	0	0	0	0	0	0	0	0	0
C85-C86	0	0	0	0	0	1	0	0	0	1
C86-C87	0	1	0	0	0	0	0	1	0	2
C87-C88	0	0	0	0	0	0	0	0	1	1
C88-C89	0	0	0	0	1	0	0	0	0	1
C89-C90	1	0	0	0	0	0	0	0	0	1
C90-C91	0	0	1	0	0	0	0	0	0	1
C91-C92	0	0	0	0	0	0	1	0	0	1
C92-C93	0	0	0	0	0	0	0	0	0	0
C93-C94	1	0	0	0	0	0	0	0	0	1
C94-C95	0	0	0	0	0	0	0	0	0	0
C95-C96	0	0	0	1	0	1	0	0	0	2
C96-C97	1	1	0	0	0	0	1	0	0	3
C97-C98	0	0	0	0	0	0	0	0	0	0
C98-C99	0	0	0	0	0	0	0	0	0	0
C99-C100	0	0	0	0	0	0	0	0	0	0
Total	9	6	5	7	5	5	10	3	4	54

Bibliography

[1] "CasperJS," casperjs.org.

[2] G. Leshed, E. M. Haber, T. Matthews, and T. Lau, "Coscripter: Automating & sharing how-to knowledge in the enterprise," in *ACM Conference on Human Factors in Computing Systems*, 2008, pp. 1719–1728. [Online]. Available: http://doi.acm.org/10.1145/1357054.1357323

[3] B. Burg, R. Bailey, A. J. Ko, and M. D. Ernst, "Interactive record/replay for web application debugging," in *ACM User Interface Software and Technology Symposium*, 2013, pp. 473–484. [Online]. Available: http://doi.acm.org/10.1145/2501988.2502050

[4] K. Sen, S. Kalasapur, T. Brutch, and S. Gibbs, "Jalangi: A selective record-replay and dynamic analysis framework for JavaScript," in *International Symposium on Foundations of Software Engineering*, 2013, pp. 488–498.

[5] J. Mickens, J. Elson, and J. Howell, "Mugshot: Deterministic capture and replay for JavaScript applications," in *USENIX Conference on Networked Systems Design and Implementation*, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855722

[6] "Sahi," sahipro.com.

[7] "The Selenium Project," http://seleniumhq.org/docs/03_webdriver.html/.

[8] T. Yeh, T.-H. Chang, and R. C. Miller, "Sikuli: Using GUI screenshots for search and automation," in *ACM User Interface Software and Technology Symposium*, 2009, pp. 183–192. [Online]. Available: http://doi.acm.org/10.1145/1622176.1622213

[9] "TestMaker Object Designer," www.pushtotest.com/designer.

[10] "Unified Functional Testing (UFT)," http://www8.hp.com/us/en/software-solutions/unified-functional-automated-testing.

[11] S. Andrica and G. Candea, "WaRR: A tool for high-fidelity web application record and replay," in *DSN*, June 2011, pp. 403–410.

[12] "Watir WebDriver," http://watirwebdriver.com.

[13] S. R. Choudhary, D. Zhao, H. Versee, and A. Orso, "WATER: Web Application TEst Repair," in *First International Workshop on End-to-End Test Script Engineering*, 2011, pp. 24–29.

[14] M. Dhatchayani, X. A. R. Arockia, P. Yogesh, and B. Zacharias, "Test case generation and reusing test cases for GUI designed with HTML," *Journal of Software*, vol. 7, no. 10, pp. 2269–2277, 2012.

[15] M. Leotta, A. Stocco, F. Ricca, and P. Tonella, "Reducing web test cases aging by means of robust xpath locators," in *IEEE International Symposium on Software Reliability Engineering Workshops*, 2014, pp. 449–454.

[16] ——, "Using multi-locators to increase the robustness of web test cases," in *IEEE International Conference on Software Testing, Verification and Validation*, 2015, pp. 1–10.

[17] R. Yandrapally, S. Thummalapenta, S. Sinha, and S. Chandra, "Robust test automation using contextual clues," in *International Symposium on Software Testing and Analysis*, 2014, pp. 304–314.

[18] M. Hammoudi, G. Rothermel, and P. Tonella, "Why do record/replay tests of web applications break?" in *International Conference on Software Testing*, 2016 (to appear).

[19] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, and D. Binkley, "Are test smells really harmful? An empirical study," *Empirical Software Engineering Journal*, vol. 20, pp. 1052–1094, 2015.

[20] B. Daniel, T. Gvero, and D. Marinov, "On test repair using symbolic execution," in *International Symposium on Software Testing and Analysis*, 2010, pp. 207–218.

[21] A. Koesnandar, S. Elbaum, G. Rothermel, L. Hochstein, K. Thomasset, and C. Scaffidi, "Using assertions to help end-user programmers create dependable web macros," in *ACM SIGSOFT Symposium on Foundations of Software Engineering*, 2008, pp. 124–134.

[22] S. Chandra, E. Torlak, S. Barman, and R. Bodik, "Angelic debugging," in *International Conference on Software Engineering*, 2011, pp. 121–130.

[23] D. Gopinath, M. Z. Malik, and S. Khurshid, "Specification-based program repair using SAT," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2011, pp. 173–188.

[24] S. Mechtaev, J. Yi, and A. Roychoudhury, "DirectFix: Looking for simple program repairs," in *International Conference on Software Engineering*, 2015, pp. 448–458.

[25] H. D. T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra, "Semfix: Program repair via semantic analysis," in *International Conference on Software Engineering*, 2013, pp. 772–781.

[26] Y. Wei, Y. Pei, C. A. Furia, L. S. Silva, S. Buckholz, B. Meyer, and A. Zeller, "Automated fixing of programs with contracts," in *International Symposium on Software Testing*, 2010, pp. 61–72.

[27] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest, "Automatically finding patches using genetic programming," in *International Conference on Software Engineering*, 2009, pp. 364–374.

[28] W. Weimer, S. Forrest, C. Le Goues, and T. Nguyen, "Automatic program repair with evolutionary computation," *Communications of the ACM*, vol. 53, no. 5, pp. 109–116, 2010.

[29] B. Daniel, V. Jagannath, D. Dig, and D. Marinov, "ReAssert: Suggesting repairs for broken unit tests," in *International Conference on Automated Software Engineering*, 2009, pp. 433–444.

[30] M. Mirzaaghaei, F. Pastore, and M. Pezze, "Supporting test suite evolution through test case adaptation," in *International Conference on Software Testing, Verification and Validation*, 2012, pp. 231–240.

[31] A. M. Memon and M. L. Soffa, "Regression testing of GUIs," in *International Symposium on Foundations of Software Engineering*, 2003.

[32] A. M. Memon, "Automatically repairing event sequence-based GUI test suites for regression testing," *ACM Transactions on Software Engineering and Methodology*, vol. 18, no. 2, pp. 4:1–4:36, 2008.

[33] S. Huang, M. B. Cohen, and A. M. Memon, "Repairing GUI test suites using a genetic algorithm," in *International Conference on Software Testing*, 2010, pp. 245–254.

[34] M. Grechanik, Q. Xie, and C. Fu, "Maintaining and evolving GUI-directed test scripts," in *International Conference on Software Engineering*, 2009, pp. 408–418.

[35] B. Daniel, Q. Luo, M. Mirzaaghaei, D. Dig, D. Marinov, and M. Pezzè, "Automated GUI refactoring and test script repair," in *First International Workshop on End-to-End Test Script Engineering*, 2011, pp. 38–41.

[36] S. Zhang, H. Lü, and M. D. Ernst, "Automatically repairing broken workflows for evolving GUI applications," in *International Symposium on Software Testing and Analysis*, 2013, pp. 45–55.

[37] N. Alshawan and M. Harman, "Automated session data repair for web application regression testing," in *International Conference on Software Testing, Verification, and Validation*, 2008, pp. 298–307.

[38] A. Stocco, M. Leotta, F. Ricca, and P. Tonella, "Why creating web page objects manually if it can be done automatically?" in *IEEE/ACM International Workshop on Automation of Software Test*, 2015, pp. 70–74.

[39] S. Bruning, S. Weissleder, and M. Malek, "A fault taxonomy for service-oriented architecture," in *IEEE High Assurance Systems Engineering Symposium*, 2007, pp. 367–368.

[40] J. H. Hayes, "Building a requirement fault taxonomy: Experiences from a NASA verification and validation research project," in *International Symposium on Software Reliability Engineering*, 2003, pp. 49–60.

[41] L. Mariani, "A fault taxonomy for component-based software," in *International Workshop on Test and Analysis of Component-Based Systems*, 2003, pp. 55–65.

[42] K. S. M. Chan, J. Bishop, J. Steyn, L. Baresi, and S. Guinea, "A fault taxonomy for web service composition," in *Service-Oriented Computing - ICSOC 2007 Workshops*, ser. Lecture Notes in Computer Science, E. Di Nitto and M. Ripeanu, Eds. Springer, 2009, vol. 4907, pp. 363–375.

[43] W. Hummer, C. Inzinger, P. Leitner, B. Satzger, and S. Dustdar, "Deriving a unified fault taxonomy for event-based systems," in *ACM International Conference on Distributed Event-Based Systems*, Jul. 2012, pp. 167–178.

[44] F. Ocariza, K. Pattabiraman, and B. Zorn, "JavaScript errors in the wild: An empirical study," in *International Symposium on Software Reliability Engineering*, 2011, pp. 100–109.

[45] F. Ocariza, K. Bajaj, K. Pattabiraman, and A. Mesbah, "An empirical study of client-side JavaScript bugs," in *International Symposium Empirical Software Engineering and Measurement*, 2013, pp. 55–64.

[46] F. Ricca and P. Tonella, "Web testing: A roadmap for empirical research," in *International Symposium on Web Site Evolution*, 2007.

[47] A. Marchetto, F. Ricca, and P. Tonella, "Empirical validation of a web fault taxonomy and its usage for fault seeding," in *International Workshop on Web Site Evolution*, 2007, pp. 31–38.

[48] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Capture-replay vs. programmable web testing: An empirical assessment during test case evolution," in *Working Conference on Reverse Engineering*, 2013, pp. 272–281.