Aldalur, I., Larrinaga, F., Perez, A. (2020). ABLA: An Algorithm for Repairing Structure-Based Locators Through Attribute Annotations. In: Huang, Z., Beek, W., Wang, H., Zhou, R., Zhang, Y. (eds) Web Information Systems Engineering – WISE 2020. WISE 2020. Lecture Notes in Computer Science(), vol 12343. Springer, Cham. https://doi.org/10.1007/978-3-030-62008-0_7

ABLA: an algorithm for repairing structure-based locators through attribute annotations

Iñigo Aldalur[1], Felix Larrinaga[1], and Alain Perez[1]

Mondragon University, Arrasate-Mondragon, Spain
ialdalur,aperez,flarrinaga@mondragon.edu

Abstract. The growth of the web has been unstoppable in the last decade, which leads to an increasing demand for extracting information from it. Apart from the need to extract information, this growth also has brought the necessity to adapt web pages to user requirements, create annotations or test web applications. Due to the evolution of web pages, the complexity of the implementation of these techniques has increased. Being able to test, annotate, adapt and extract information from web pages correctly and efficiently has become a primary task. In order to perform all these tasks, it is mandatory to have the best mechanisms to effectively and unequivocally locate the desired elements throughout the web page life cycle, especially when a web page evolves. Different mechanisms are used to find web nodes. These mechanisms, called locators, are prone to fail over time owing to changes on websites. Many authors improve life expectancy of locators developing algorithms that use different types of locators. Some others have created algorithms that regenerate locators by saving extra information from the previous structure of the website. These algorithms extend the useful life of locators but their computational and storage cost is much higher. To avoid these problems, we have designed an algorithm that employs an attribute system embedded in the HTML code. The algorithm is able to regenerate the locators based on these attributes every time a single change takes place in a given element attribute. The evaluation of the proposal shows a much lower computational cost than in previous works.

Keywords: Locators · Annotation · Mining · Testing · Web Attributes

1 Introduction

Since the creation of the web 30 years ago, one of the objectives has been the extraction of relevant information from this universal source for different purposes. To automatically extract this information locators are used. "A Web locator is a mechanism for uniquely identifying an element on the Web Content i.e. in the Document Object Model (DOM)" [1]. Locators have evolved over time from being fragile and fail to minimal changes of the website to tolerate changes. However, these changes continue to cause locators to stop working, leading to high maintenance costs in terms of both time and money.

Different types of locators have been developed over time. The first ones [22] were based on coordinates and were prone to change at the slightest change. For this reason they have became obsolete. The second generation is based on the structure of the web page (the DOM). These locators are more robust than the previous ones but as soon as there are changes in the structure, they tend to stop working. [22] classifies structure-based locators into XPath locator and attribute locators. Finally, a third generation of image-based locators has been developed (visual locators). These locators use image recognition techniques in order to find a certain element on a web page [22]. Moreover, [1] adds content-based locators to the list. Content-based locators work with the text composition of a website. These locators extract text content even when the desired text is in the middle of a paragraph thanks to its ability to select data within text node content. It is mainly used for web annotation and web harvesting, which is the process of automatically collecting information from the Web [10].

Locators are used in different scenarios: web annotation (establishes a relation between two resources [11]); web augmentation (is the alteration of the original user interface, generally by using scripts running at the client side [12]); web automation (is an activity that can be performed without human intervention [18]); data extraction (retrieves information stored on the Web [10]); testing (are processes designed to verify that a computer code works correctly [17]). For each of these cases you can use different types of locators. Despite the fact that coordinate based locators have almost ceased to be used, some studies still use them for testing and automation [4]. Coordinate-based locators are appropriate when the locations of the tested or automated website nodes are stable. In other words, when nodes are kept in the same position. It is important to highlight that coordinate locators can only be used to locate leaf nodes. Visual locators are appropriate when the structure of the website changes frequently and, as a consequence, structure based locators are less suitable. Additionally, if nodes are relocated on the website, coordinate based locators are useless and visual locators are able to find the element wherever the node is. Hence, visual locators are also suitable for testing and automation [27]. Content based locators are appropriate when the desired web information is in the text [23]. Other types of locators are unable to extract information from a text but structure based locators can be used when the data extract or annotation is the whole node. Visual locators are not suitable because content is not the same all the time. Structure based locator are appropriate for all scenarios [7].

Not only are structure based locators the most extended because of their usefulness in all scenarios, but also they are the most robust type of locator. [21] compares structure based locators to visual locators and concludes that structure based locators are more robust. Furthermore, they compare attribute structure based locators with XPath concluding that the ID attribute is the most robust of all, whereas XPath is the least robust. Taking into account previous works done to compare different types of locators, [1,21] summarize locator robustness classification as follows: structure-ID, structure-attributes, structure-XPath, visual, content, coordinate.

The remainder of the paper is organized as follows. Section 2 analyses the problem we want to solve, its causes and consequences. Section 3 discusses related work in order to give the reader an idea of what has been done in this area. Section 4 describes our approach to solve or mitigate the previously explained problem. Section 5 presents the validation of our approach and its results. Finally, Section 6 shows features we would like to enhance about our solution and it concludes the paper.

2 Problem analysis

The problem

Different approaches successfully use locators in their daily routines. Nonetheless, after websites upgrades, previously selected web nodes will no longer be found by standard locators. No matter how robust the locators are, they will eventually fail because all websites are updated over time [21].

Causes

One of the main causes is locator fragility. There are several works that have calculated the fragility of the locators [4, 9, 22]. This fragility is due to the fact that web pages are frequently updated because they are not able to meet users' needs [20]. [16] studied some reasons why locators stop working. The conclusions are that updates on the web may be due to changes in the structure (the DOM), appearance (CSS styles) or content. For example, changes in structure affect locators based on structure, those based on coordinates if the change implies a change of location and those based on content if the change affects the order of the text. The visual locators are those that better support the changes in the structure since they are not affected by these changes. When changes are implemented in styles, the most affected locators are visual and coordinate-based. Conversely, those based on structure and content are not impacted by this type of change since content and structure remain exactly the same as before. Finally, if what changes in a web page is the content, the most affected will be the content-based locators.

Consequences

When a locator breaks, it has to be fixed. Additionally, when there are changes in a website, the more locators there are the more locators will stop working. Therefore, fixing them is time-consuming, which results in high economic cost [14]. That cost rises if such changes are frequent. It is important to emphasize that many of these locator updates are not done automatically but manually, leading to a higher cost [6]. If a locator causes an application to stop working, users could perceive the application as low quality and therefore refuse to continue using it. Therefore, fixing locators and keeping them active is of vital importance. Our goal is to reduce the number of times a developer has to fix failed locators, thus reducing the economic cost and increasing user loyalty.

3 Related work

First locator implementations focus mainly on generating robust algorithms. Robustness is defined as the ability of a computer system to cope with errors during execution [1]. A good example of development of robust web locators can be found in Robula+ [22]. Robula+ is an algorithm for the automatic creation of structure locators, more precisely XPath expressions. This algorithm prioritizes attributes by recognizing their ability to create robust XPath locators (ID and class attributes). Another similar example is [24] that shows that they can create a robust extractor for product information extraction by combining supervised classification with unsupervised methods. [3] claims that generating locators manually is demanding due to the dynamic nature of the DOM since it involves selecting multiple DOM elements. That is why they have created an automated technique for synthesizing DOM element locators using examples provided interactively by the developer. In spite of the fact that these approaches improve the robustness of previous researches, they end up failing.

In order to extend the useful life of locators, other authors propose the use of multiple locators. [2] uses Floper, a tool developed by their research group that generates XPath expressions that can be adapted manually to obtain a set of alternative queries. In essence, this means that the applications have more than a unique XPath expression to find the desired node. If a locator fails, another one will be utilized to retrieve the same result. [5] presents a web test generation algorithm that selects the most promising alternatives creating more than one locator for each element. [28] uses numerous parallel threads in order to evaluate different XPaths. Nevertheless, even if all these approaches extend life expectancy of locators, they wind up failing.

In recent times, research has changed the approach by trying to regenerate a locator when it failed. There are numerous studies which use XPath expressions to locate UI controls. [25] identifies failed tests and no longer valid XPath expressions. Then, it uses invalid XPath expressions and compares the two DOMs corresponding to a view in the new version and on the original page. Finally, it uses the DOM difference to repair the XPath expressions. In a similar manner, [9] developed a method of automatic wrapper reparation. It is based on comparing stored original structural DOM information for the Web wrapper with the new version and searching for similarities. [26] carries out a visual technique for web testing algorithms, including visual-based test repair for migrating DOM-based tests to visual tests. [15] repairs an invalid locator by using the attributes from the old version of the node. All these articles, in a nutshell, compare previous DOM with the current DOM to regenerate the locator. Nonetheless, it is not the unique technique because other authors use previous node attributes to recreate the new locator. For example, [1] regenerates XPath expressions when they are invalid owing to the fact that they save all structure data of the node and all its ancestors. If there is no single element obtained, then it tries all combinations of ancestor's structure information of the old element gradually. If the created new XPath expression does not locate a unique element using its own attribute combinations, the attribute combinations of the ancestor node of this element are utilized. This process is repeated until a single node is detected. It is considered a failure if no unique node is located when all the ancestor structure data is used. [8] creates a locator based on thirteen attributes and when the locator fails, it uses previously stored node's attributes in order to create a new locator that is able to to find the new node. [19] affirms that some "methods tried to repair broken locators by using structural clues, but these approaches usually cannot handle radical changes to page layouts". They determine which node properties (attributes, texts, images, and positions) are trustworthy after comparing old and the new version to create a new locator. Finally, some other works have a manual component when they repair the locator. SITAR [13] is a system to interactively repair test scripts. This reparation is completed by mapping the new version of the interface and creating manually the new version. Unfortunately, all these algorithms are not always able to regenerate the locator despite the fact their performance prolongs locator life. In the next section we present an approach that prolongs locator efficacy in all the above scenarios considering that locators are run frequently.

4 Repairing structure attribute based locators

Despite all the efforts that have been made to develop the most robust locators, with the passage of time locators end up failing. As we have seen in section 3, certain authors use several of these algorithms in their applications and thus extend locators' useful life. Most recent works try to regenerate broken locators by comparing the old and the new DOM, by using the old attributes. In our research group we have developed numerous web applications for different projects which we test every time we make a change. The problem arises when certain changes made some of the locators used in the test stop working. Then, we had to dedicate time to locate them and generate the test again causing a damage in time and money. Therefore, based on the work of restoring locators, we have developed an algorithm that supported by specific attributes in the DOM, it is able to regenerate the locators based on structure effectively, Annotation-Based Locator Algorithm (ABLA).
To achieve our goal we have defined four different attributes that are used to tag nodes. When the node id, class or another attribute is changed, these especial attributes are used to tag nodes maintaining the previous values:

- **prevId**: used when the id value is changed.
- **prevClass**: used when the class value is modified.
- **preva**: this is used when the value is changed maintaining the attribute name. The attribute name is concatenated to this tag.
- **prevv**: the value name is concatenated to this tag. This is used when the attribute name is changed maintaining the value.

The ABLA algorithm capable of testing a website and restoring locators in case they are broken is shown in Algorithm 1 in which the pseudo code of the algorithm is outlined.

Algorithm 1 performs according to the following steps of the procedure:

Algorithm 1 ABLA algorithm

```
 1: for locatorList do
 2:     attribute = getLocatorAttribute(locatorList, i, 0)
 3:     value = getLocatorValue(locatorList, i, 1)
 4:     node = executeLocator(attribute, value)
 5:     if node != null then
 6:         executeTest(node)
 7:     else
 8:         newNode = findNewNode(attribute, value)
 9:         if newNode != null then
10:             newNodeAttr = getNewNodeAttr(newNode)
11:             newNodeValue = getNewNodeValue(newNode)
12:             restoreLocator(locatorList, attribute, newNodeValue, i)
13:             executeTest(newNode)
```

– Line 1: We have a list initialized with all nodes we want to find in the website.
– Lines 2 and 3: All attribute types and their values are read from the list one by one. There are three different types of attributes: id, class and the rest.
– Line 4: Each type of attribute has a different mechanism to find the node and this is the reason why we classify them in three categories. ”executeLocator” function executes this three different locators in order to find the correct node.
– Line 6: If a node is found, this means that the locator is still correct and the test is executed normally.
– Line 8: Otherwise, ”findNewNode” function tries to find the desired node. This function will be explained in algorithm 2. If the function does not find any node, the test fails and it must be restored manually.
– Line 10 and 11: If a node is found, the new attribute and value are extracted. If the attribute is an id or a class, these attributes will be maintained and what will change is their value. If it is another type of attribute, the change could have occurred in the name of the attribute or its value.
– Line 12: the new attribute and the new values are stored in the list replacing the old values and attributes.
– Line 13: the test is executed.

In line 8 of the Algorithm 1 the function ”findNewNode” is executed. It needs the attribute type and its value to find the desired node. The ”findNewNode” function is explained in pseudo code in Algorithm 2. Algorithm 2 performs the following step by step procedure:

– Line 1 and 2: the function needs two parameters, the attribute type and its value to carry out its action properly..
– Line 4 and 5: If the attribute is id, the function searches the attribute ”prevId” and the value the previous id had. If the ”findNode” function does not find any node with this attribute and value, the function will return null. On the contrary, if the ”findNode” function finds a node, this element will be returned.

– Line 6 and 7: If the attribute is class, the function looks for the attribute "prevClass" and the value of the previous class. In the same manner, the "findNode" function will return null or it will return the found element.
– Line 9: If the attribute is different from id or class, the function looks for the attribute "preva" concatenating the previous attribute name and the value of the previous attribute. This is because we firstly check if the value has been updated and the attribute is maintained as in the preceding if sentences.
– Line 10 and 11: If the previous step does not find a single node, the "findNode" function checks if the attribute name has been upgraded instead of the value.
– Line 12: the function returns the updated node or null in case no element has been found with the previous attributes and values.

It is also very important to emphasize that the solution of class attributes and the rest of attributes must be unique. If the function provides more than one node it means that it is no longer unique and null will be returned. Since id attributes are unique, they do not need this check.

Algorithm 2 findNewNode function

```
1: param1: attribute
2: param2: value
3: if attribute == "id" then
4:     node = findNode("prevId", value)
5: else if attribute == "class" then
6:     node = findNode("prevClass", value)
7: else
8:     node = findNode("preva"+attribute, value)
9:     if node == null then
10:        node = findNode(attribute, "prevv"+value)
11: return node
```

It is very important to highlight that this method not only serves for small changes that occur continuously in the development of web applications. This method enables, based on annotations, to run test, perform augmentation or data extraction or data mining from web pages that have been completely changed. This can be carried out if we include the old attributes in the new version. The new nodes that have the same representation or aim in the old version website must include previous id, class and other attribute values.
With this work we intend to propose labels which can be created automatically with an editor each time the developer changes any of the attributes. If the creation of these tags in the code is automatically generated, the cost will be practically null for the developer. These labels will be maintained for as long as the developer changes the value of the same attribute when the value of the proposed labels will be updated. This method diminishes in its totality the need to store the information. Unlike [25] that keep the previous version of the

DOM or [1] that store all the attributes of the ancestors. In addition, the cost of computation is much lower. This aspect is demonstrated in the following section.

5 Validation

We have defined the following research questions to evaluate our approach:
RQ1 (Efficacy): How effective is ABLA at restoring broken locators?
RQ2 (Efficiency): What is the run-time of ABLA and what is the difference between ABLA and other approaches?

5.1 Approach and results

The aim of the RQ1 is to evaluate the efficacy of ABLA in different websites. 5 different websites have been utilized to evaluate the algorithm. The criteria to select these websites has been their number of nodes, the number of attributes and their structure complexity which is illustrated in table 1. The complexity of the HTML structure defines the number of sub-levels existing in the HTML or how far elements are settled down in the structure. We have simulated the behaviour of a web developer. We have identified the nodes that have changed their attribute values and we have added previous attribute values to proposed attributes. WayBackMachine[1] has been applied to obtain old version of the evaluated web pages. The versions of the first day of each month in 2019 has been used for comparison. In this evaluation hidden nodes and removed nodes have not been taken into consideration.
Table 2 presents the results of the evaluation. The first column shows the web-

Table 1. A comparison between websites used to test ABLA

Website	#Elements per page	#Attributes per element	Type of structure
IMDB	Many	Many	Moderate
Wikipedia	Moderate	Moderate	Simple
Apple	Moderate	Many	Moderate
Firefox	Few	Few	Simple
BBC	Many	Moderate	Complex

sites tested. The next three illustrate the broken IDs, classes and other attributes (first value in each cell) and the number of restored locators by ABLA (second value in the cell). The last column shows the percentage of restored locator in each website. This column shows how ABLA is able to restore always the broken locator and find the desired node even if the website has been updated.
With regard to RQ2, the goal is compare its runtime with other algorithms that restore broken locators too. Besides, we want to evaluate the efficiency of our approach.
Figure 1 compares the execution time of ABLA with other algorithms that regenerate broken locators. In particular the same 32 nodes from the same web

[1] https://archive.org/web/

Table 2. ABLA repair results

Website	Broken/ Restored IDs	Broken/ Restored Classes	Broken/ Restored Attributes	%Restored locators
IMDB	12/12	28/28	267/267	100%
Wikipedia	5/5	3/3	115/115	100%
Apple	0/0	145/145	235/235	100%
Firefox	0/0	6/6	22/22	100%
BBC	1/1	95/95	86/86	100%

Table 3. Regenerate locator algorithms execution order

Algorithm	Exec. order	Success	Test dependency
ABLA	$O(n)$	100%	n: nodes
[25]	$O(m^n)$	87.7%	m: test, n: node children
Weighted Tree Matching [9]	$O(m^n)$	90%	m: test, n: node children and siblings
VISTA [26]	$O(m^n)$	81%	m: test, n: images
WATERFALL [15]	$O(m^n)$	89.3%	m: test, n: node attributes
Regenerative locator [1]	$O(m^n)$	73.3%	m: test, n: node attributes
Genetic Algorithm [8]	$O(m^n)$	87%	m: test, n: node attributes
COLOR [19]	$O(m^n)$	77-93%	m: test, n: nodes or attributes
SITAR [13]	$O(m^n)$	41-89%	m: test, n: nodes

page where extracted and the time needed to restore each locator was calculated. Answering the research question 2, ABLA has been evaluated individually and it needs 32ms to extract 32 nodes. Furthermore, Regenerative locator [1], VISTA [26], WATERFALL [15], COLOR [19] and SITAR [13] have been evaluated obtaining similar results as those presented in their corresponding researches. Table 3 illustrates the execution order of each algorithm. Their execution order is elevated because all previous algorithms are based on the number of test and the number of nodes, attributes or node siblings and what is more, complex node selection to extract optimal options. All this makes that their execution time is higher comparing with ABLA (see figure 1). In case of Regenerative locator [1], a simpler algorithm is used and its execution time is similar to ABLA even if it needs around 100ms to restore each locator. Table 3 also shows the performance of each algorithm. Regenerative locator presents the lowest time cost. In general, these algorithms are excellent restoring their locators with a higher success than 80% but ABLA is superior.

5.2 Limitations

All previous work mentioned in the related work obtain excellent results but all of them have limitations because when certain conditions are met, they all end up failing. In the case of ABLA, if the developer has carried out two updates between two executions on the same node in the same attribute, the algorithm will not find the web node. Given that attribute-based locators are the most

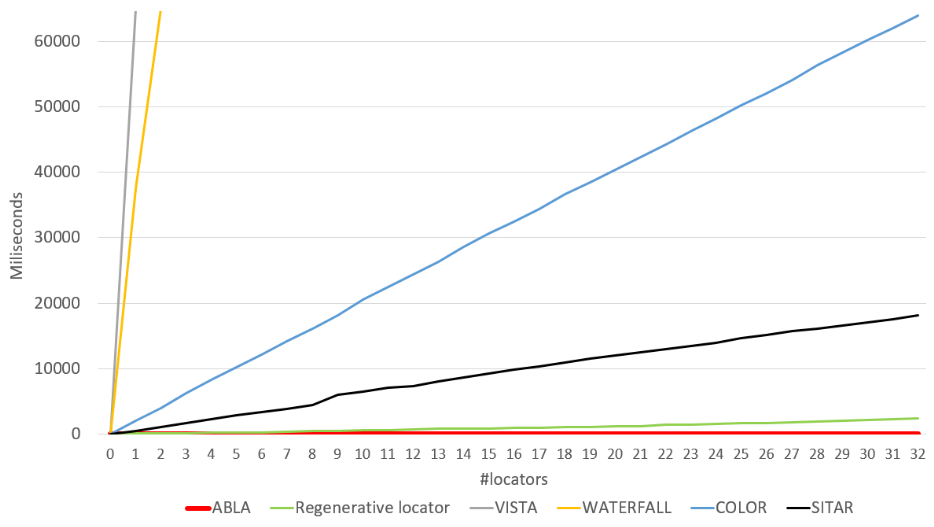

Fig. 1. Comparison between algorithms that regenerate locators

robust since they are more resilient against web upgrades, it is very unlikely that they will be updated more than once in a short period of time. To justify this statement, note that in [22] they concluded that in 6 years, less than 2% of the ID-attributes and 18% of the other attributes had failed.
Another limitation is that this regeneration algorithm is intended to be applied on attributes that are unique. Therefore, it cannot be applied to all nodes of a web page since not all nodes have unique attributes.

6 Future work and conclusions

Locators are sensitive to changes that occur on websites. All developed methods have tried to generate locators as robust as possible and although they are able to find the desired node for a longer time, they tend to fail in the long term when many changes in a page take place. For this reason, recent work tries to regenerate the locators that have stopped working by saving information related to the old code of the web page. All these algorithms are much more robust but they are less efficient because they require a much higher computational cost. Moreover, they need additional memory space to store information related to old versions of the website. The method presented in this article allows to save that extra storage since the data is inserted using special attributes in the HTML code. In addition to this, the algorithm is more efficient since it is not necessary to test a large number of different options until a locator that works is found. If the main locator fails, ABLA only requires to apply another execution, thus considerably reducing regeneration time. Finally, the efficacy of the algorithm reaches 100% if it is executed every time there is a change in the HTML code.
In our future work we plan to investigate and develop an algorithm that is able to regenerate XPaths by using these special tags. XPaths are applicable to all

the nodes of a web page to find a specific node. As a consequence, we would avoid one of the limitations of this work since the ABLA algorithm only applies to nodes with unique attributes. In addition to this, our goal is to develop simple mechanisms for detecting updates on third-party websites and be able to keep a 100% effectiveness.

Acknowledgments This work was carried out by the Software and Systems Engineering research group of Mondragon Unibertsitatea (IT1326-19), supported by the Department of Education, Universities and Research of the Basque Government.

References

1. Aldalur, I., Díaz, O.: Addressing web locator fragility: a case for browser extensions. In: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2017, Lisbon, Portugal, June 26-29. pp. 45–50 (2017)
2. Almendros-Jiménez, J.M., Tedesqui, A.L., Moreno, G.: Annotating "fuzzy chance degrees" when debugging xpath queries. In: Advances in Computational Intelligence - 12th International Work-Conference on Artificial Neural Networks, IWANN 2013, Puerto de la Cruz, Tenerife, Spain, June 12-14. pp. 300–311 (2013)
3. Bajaj, K., Pattabiraman, K., Mesbah, A.: Synthesizing web element locators (T). In: 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13. pp. 331–341 (2015)
4. Bartoli, A., Medvet, E., Mauri, M.: Recording and replaying navigations on AJAX web sites. In: Web Engineering - 12th International Conference, ICWE 2012, Berlin, Germany, July 23-27. pp. 370–377 (2012)
5. Biagiola, M., Stocco, A., Ricca, F., Tonella, P.: Diversity-based web test generation. In: 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering ESEC/FSE 2019 August 26–30, Tallinn, Estonia. pp. 231–242 (2019)
6. Bures, M., Filipsky, M.: Smartdriver: Extension of selenium webdriver to create more efficient automated tests. In: 6th International Conference on IT Convergence and Security, ICITCS 2016, Prague, Czech Republic, September 26. pp. 1–4 (2016)
7. Chang, C., Lin, Y., Lin, K., Kayed, M.: Page-level wrapper verification for unsupervised web data extraction. In: Web Information Systems Engineering - WISE 2013 - 14th International Conference, Nanjing, China, October 13-15. pp. 454–467 (2013)
8. Eladawy, H.M., Mohamed, A.E., Salem, S.A.: A new algorithm for repairing weblocators using optimization techniques. In: 13th International Conference on Computer Engineering and Systems (ICCES). pp. 327–331 (Dec 2018)
9. Ferrara, E., Baumgartner, R.: Intelligent self-repairable web wrappers. In: AI*IA 2011: Artificial Intelligence Around Man and Beyond - XIIth International Conference of the Italian Association for Artificial Intelligence, Palermo, Italy, September 15-17. pp. 274–285 (2011)
10. Ferrara, E., Meo, P.D., Fiumara, G., Baumgartner, R.: Web data extraction, applications and techniques: A survey. Knowl.-Based Syst. **70**, 301–323 (2014)
11. Fiorelli, M., Pazienza, M.T., Stellato, A.: A flexible approach to semantic annotation systems for web content. Int. Syst. in Accounting, Finance and Management **22**(1), 65–79 (2015)

12. Firmenich, D., Firmenich, S., Rivero, J.M., Antonelli, L., Rossi, G.: Crowdmock: an approach for defining and evolving web augmentation requirements. Requir. Eng. **23**(1), 33–61 (2018)
13. Gao, Z., Chen, Z., Zou, Y., Memon, A.M.: SITAR: GUI test script repair. IEEE Trans. Software Eng. **42**(2), 170–186 (2016)
14. Guo, J.: Reducing human effort in web data extraction. Ph.D. thesis, University of Oxford, UK (2017)
15. Hammoudi, M., Rothermel, G., Stocco, A.: WATERFALL: an incremental approach for repairing record-replay tests of web applications. In: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18. pp. 751–762 (2016)
16. Hammoudi, M., Rothermel, G., Tonella, P.: Why do record/replay tests of web applications break? In: IEEE International Conference on Software Testing, Verification and Validation, ICST 2016, Chicago, USA, April 11-15. pp. 180–190 (2016)
17. Herbold, S., Bünting, U., Grabowski, J., Waack, S.: Deployable capture/replay supported by internal messages. Advances in Computers **85**, 327–367 (2012)
18. Huizinga, D., Kolawa, A.: Automated Defect Prevention: Best Practices in Software Management (2007)
19. Kirinuki, H., Tanno, H., Natsukawa, K.: COLOR: correct locator recommender for broken test scripts using various clues in web application. In: 26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27. pp. 310–320 (2019)
20. Lee, T.Y., Bederson, B.B.: Give the people what they want: studying end-user needs for enhancing the web. PeerJ Computer Science **2**, e91 (2016)
21. Leotta, M., Clerissi, D., Ricca, F., Tonella, P.: Visual vs. dom-based web locators: An empirical study. In: Web Engineering, 14th International Conference, ICWE 2014, Toulouse, France, July 1-4. pp. 322–340 (2014)
22. Leotta, M., Stocco, A., Ricca, F., Tonella, P.: Robula+: an algorithm for generating robust xpath locators for web testing. Journal of Software: Evolution and Process **28**(3), 177–204 (2016)
23. Lin, A.Y., Ford, J., Adar, E., Hecht, B.J.: Vizbywiki: Mining data visualizations from the web to enrich news articles. In: Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27. pp. 873–882 (2018)
24. Potvin, B., Villemaire, R.: Robust web data extraction based on unsupervised visual validation. In: Intelligent Information and Database Systems - 11th Asian Conference, ACIIDS 2019, Yogyakarta, Indonesia, April 8-11. pp. 77–89 (2019)
25. Song, F., Xu, Z., Xu, F.: An xpath-based approach to reusing test scripts for android applications. In: 14th Web Information Systems and Applications Conference, WISA 2017, Liuzhou, China, November 11-12. pp. 143–148 (2017)
26. Stocco, A., Yandrapally, R., Mesbah, A.: Visual web test repair. In: Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09. pp. 503–514 (2018)
27. Yeh, T., Chang, T., Miller, R.C.: Sikuli: using GUI screenshots for search and automation. In: Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology, Victoria, BC, Canada, October 4-7. pp. 183–192 (2009)
28. Zhang, Y., Pan, Y., Chiu, K.: A parallel xpath engine based on concurrent NFA execution. In: 16th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2010, Shanghai, China, December 8-10. pp. 314–321 (2010)