Capture and Replay Tools in the Field

Deepa Chhetri*, Brian Pollack†

Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, OH 44106
*dxc496@case.edu, †bmp55@case.edu

***Abstract*—We compared four different capture and replay tools, mainly based on their performance and ease of use. We compared Reanimator, Sahi, Telerik, and Selenium IDE and determined that Sahi is overall the best capture and replay tool. Additionally, we extended Reanimator to fix core bugs, upgrade it to support JQuery 3.2.0 (the current version at time of writing), and enable it to upload its captures to a server. We believe that our extended version of Reanimator is now useful to JavaScript developers and can conveniently be used in the field.**

Keywords
GUI testing; capture and replay tools; Reanimator

I. Introduction

SOFTWARE testing is a process of evaluating software for its functionality by performing some test cases on an attribute or capability of a program or system and determining that it meets its required results. Testing involves the execution of a software component or system component to evaluate one or more properties of interest. Testing the graphical user interface (GUI) of a software product is important to ensure the quality of the system and therefore to improve the user experience[5]. It not only involves testing the GUI but also the functionality of system. But manual testing the GUI is tedious and time consuming. GUI capture and replay tools were developed as a mechanism for testing the correctness of interactive applications with graphical user interfaces. Using a capture and replay tool, a tester can run an application and record the entire interactive session. The tool records all the user's events, such as the keys pressed or the mouse movements, in a log file called test scripts. Given tets scripts, the tool can then automatically replay the exact same interactive session any number of times without requiring a human user. This helps in early detection of bugs that might lead to system failure and therefore speed up the process of development.

The advantages of using capture and replay tools include saving time by making the process quicker and easier. Capture and replay tools save time by not forcing the developers to hand-code test scripts which would need to be updated each time the application is changed. They also make it simple for testers without significant programming skills to automate their test cases [3]. This means that companies can hire employees dedicated to testing who are not required to know high level programming, therefore lowering testing costs. However, these tools do have some drawbacks. They don't generate test cases automatically, sometimes the recorded test scripts are inefficient and require manual intervention, and the tools know only how to interact with widgets, but they have no idea with which application they are operating [8]. During regression testing, any change to the GUI may cause a test case to break or stop execution of the updated version [3].

In this paper we compare four capture and replay tools using different criteria. This comparison study will be beneficial to software developers who want to choose a capture and replay tool which would be a good fit for testing their applications. Further we explain how we extended Reanimator, an existing capture and replay tool, and enhanced it by adding functionality that will help web developers effectively run capture and replay tests in the field.

The remainder of this paper is organized as follows: Section 2 discusses previous research regarding the comparison of capture and replay tools; Section 3 summarizes each tool that was used for our comparison; Section 4 decribes the evaluation

criteria used to establish the comparison; Section 5 describes the results of our analysis; Section 6 describes our modifications to Reanimator; Section 7 contains a summary of our work.

II. Literature Review

Nedyalkova and Bernardino [5] established a comparison of capture and replay tools. They evaluated five open source capture and replay tools: Abbot, Jacareto, JFCUnit, Marathon, and Pounder. The criteria that they used for comparison is divided into two major groups - "General Characteristics" and "Capture/Replay Characteristics". General characteristics include:

1) Easy to Install
2) User Interface
3) Easy to Use
4) Easy to Launch Application Under Test (AUT)
5) Documentation
6) Tutorials
7) Examples
8) Programming Skills Required

Capture/Replay Characteristics include:

1) Text Field - interaction with a JTextField
2) Mouse Move - detection of mouse movements on a component
3) Mouse Drag - detection of mouse drags
4) Mouse Clicks - detection of mouse clicks
5) Component - interaction with a window
6) Scrolling - interaction with the scrollbar of a JTextArea
7) File Dialog - navigating the directories in a JFileChooser
8) Combo Box - selecting an item in a JComboBox

Based on their analysis, Nedyalkova and Bernardino determined that Jacareto is the best tool among the five they evaluated. Jacareto is easy to install, had better documentation, does not require additional programming skills, and supports all capture and replay characteristics except for navigation of dialog box. Jacareto allows the tester to speed up or slow down the replay of the application log, helping make testing more efficient.

Similarly, Rodrigues et al. [7] performed an empirical comparison to evaluate Capture/Replay (CR based) and Model-Based Performance (MBT) testing tools. They studied the effort required (time spent) to use both CR-based and MBT tools. They compared LoadRunner and Visual Studio, both CR-based tools, and the PLeTsPerf MBT tool to create performance test scripts and scenarios to test web applications. LoadRunner is a performance testing tool based on the CR technique that supports scripts generation and execution to test Web-based applications. Visual Studio is an IDE developed by Microsoft to support software design, development and test. PLeTsPerf is a model-based performance testing tool. Rodrigues et al. used used an in-vitro approach, since it refers to the experiment in the laboratory under controlled conditions, addressing a real problem, i.e., the differences in individual effort to create performance test scripts and scenarios using LoadRunner,VisualStudio and PLeTsPerf. They performed the experiment to answer three research questions:

1) What is the effort to generate a single performance test script and scenario using PLeTsPerf, LoadRunner, and VisualStudio?
2) What is the effort to re-generate performance test scripts and scenarios when using PLeTsPerf, LoadRunner, and VisualStudio?
3) What is the effort to generate a set of performance test scripts and scenarios using PLeTsPerf, LoadRunner, and VisualStudio?

The experiment execution phase was split into three sessions, each composed of three tasks. These tasks were performed using the PLeTsPerf, LoadRunner and VisualStudio approaches, one for each task, and all tasks should generate an equivalent performance test script as an output. They summarize the results by giving average time spent per treatment to complete each session. They determined that for simple testing tasks the effort of using CR-based tool was less than MBT tool but as the complexity of testing task increases the advantage of using MBT grows significantly.

Mugshot [4] is a deterministic capture and replay tool developed by Microsoft. It captures every event that is executed in JavaScript program which can be replayed deterministically at any later time to analyze the sequence of events that might have led to system failure. To capture application activity, it records all sources of nondeterminism. If an application is run again and injected with the same non deterministic events, the program will follow the same execution path that was observed at logging

time. It has its client side implemented in JavaScript to easily record logs instead of downloading additional plug-ins. For each new event it captures, it creates a log entry that contains a sequence number and wall clock time. The entry also contains the event type and enough type-specific data to recreate event at play time. Mugshot uses a caching proxy to reproduce the load events in the log. This is the limitation of Mugshot. If an application fetches external contents that does not pass through the proxy, Mugshot cannot guarantee faithful replay of its data or its load time.

Microsoft calls JARDIS a "time-traveling" debugger for JavaScript. It is part of their ChakraCore JavaScript engine and a fork of the Node.js runtime environment [2]. The overall system design is a record-replay-snapshot architecture. The snapshot algorithm is based on a tool called Tardis with the record-replay work done in the JsRT hosting API layer that Node.js uses to interface with the JavaScript runtime [1]. It provides a rich set of functionality for recording and navigating the execution history of a program including:

1. Local Debuging and Offline Recording : JARDIS also supports postmortem debugging. In this scenario, the developer runs their application under JARDISs record mode when deployed to their users, given their consent. When an error occurs, the execution trace is sent to the specified location for debugging later. The collected trace has sufficient information to replay the execution of the program exactly, including any non-determinism, and show the value of any JavaScript program variable that the developer might be interested in. Thus, the debugging experience is the same as if the developer had a debugger attached to the remote Node.js process when the trace was collected.
2. Reverse-Step Local and Dynamic operations : It allows the developer to step-back in time to the previously executed statement in the current function or to step-back in time to the previously executed statement in any frame including exception throws or callee returns. There are two types of reverse-step opertaion. The first type of reverse-step operation is a reverse-step (rs) to the previously executed line in the current call frame. This is simply the reverse version of the forward step provided by existing graphical and command line debuggers. The second type of reverse operation is reverse-step dynamic (rsd) which acts as the reverse version of step-into operations in existing debuggers.This operation steps-back in time to the previously executed statement in any frame.
3. Reverse to Callback Origin operation : It reverses time to the line that registered the currently executing callback e.g. the call to setTimeout where the currently executing function was registered.

III. Description of Capture and Replay Tools Used

In this section, we give a short description of capture and replay tools analyzed in this study. We focused on those capture and replay tools that are built for internet applications. Here, we compared four tools − Reanimator, Sahi, Telerik and Selenium IDE:

A. Reanimator

Reanimator[1] captures non-deterministic input to a JavaScript application in a log that can be replayed at a later time. It was originally designed for recording web application crashes in the wild for later debugging. It is inspired by Microsofts Mugshot. It is built on JQuery 1.8.3. The code for Reanimator is available on GitHub and is under MIT license. It works only on Firefox. It supports both Windows and Linux platforms.Reanimator consists of a core, which is responsible for capture setup and driving the replay, and one or more plugins, which are responsible for capturing and replaying non-deterministic input. Reanimator ships with plugins for capturing and replaying random numbers, dates, and timer interrupts.

B. Sahi

Sahi[2] is an Automation testing tool that is used for testing web application. There are two versions of Sahi tool. One is open source and is available on SourceForge. The other, named Sahi Pro, is proprietary and is available on their official website. It has powerful abilities for recording and replaying across browsers; different language drivers for writing test scripts (Java, Ruby); and support for AJAX and highly dynamic web applications. Sahi also

[1]https://github.com/WaterfallEngineering/reanimator
[2]http://sahipro.com/

supports https and NTLM authentications. It works on Internet Explorer, Firefox and Chrome. It only supports Windows platform. The following figure shows how Sahi fits in a simulated user operation:

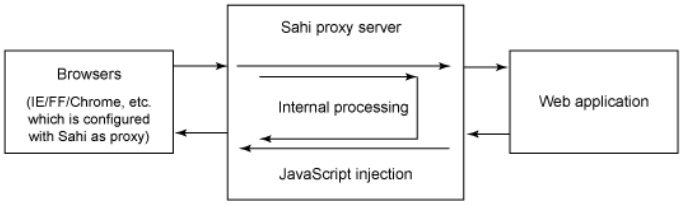

Fig. 1: Simulated user operation

C. Telerik

Telerik Test Studio[3] is commercial windows-based software testing tool with Visual Studio plug-ins. It facilitates web and desktop (GUI) functional testing, performance testing and mobile app testing with Record and Replay features. It supports JavaScript, HTML, ASP.NET, Ajax, Silverlight etc and facilitates quick validations. It works on Internet Explorer, Firefox, Chrome and Safari. It has features like visual storyboard and 3D element selection. It also offers script-lesstest automation for Silverlight applications. Test Studio offers three product versions:

1) The Functional version performs web and WPF testing and includes the Visual Studio plugin.
2) The Load version performs load testing.
3) The Ultimate version combines Web, Mobile, WPF, Load testing and Test Studio for APIs.

D. Selenium IDE

Selenium IDE[4]: Selenium is known as umbrella project that enables web browser testing for all browsers. It is a open source free application supports GUI Testing and web functional testing. It is implemented as a Firefox extension, and allows recording, editing, and debugging tests. It includes the entire Selenium Core which allows easy and quick record and play back tests in the actual environment that they will run in. It supports both Windows and Linux platforms.Selenium IDE supports autocomplete mode when creating tests. This feature serves two purposes:

- It helps the tester to enter commands more quickly.
- It restricts the user from entering invalid commands.

[3]http://www.telerik.com/teststudio

[4]http://www.seleniumhq.org/projects/ide/

IV. Evaluation of Existing Tools

The four tools discussed in the previous section were installed on an HP laptop with Intel core i5 processor (2.30 GHz), 4 GB RAM and windows 8.1 operating system with Java 1.8. We installed the tools Sahi, Telerik and Selenium. The code for Reanimator was directly deployed on a web server we manage. Initially, to test these tools (except for Reanimator) a simple online calculator written in Javascript was deployed to one of our webservers. The demo code for Reanimator comes with a simple puzzle game that we used for its testing.

We divided the criteria on which these tools are to be evaluated in three categories -

1. Evaluation for ease of deployment - In this category, we tried to evaluate the tools on the basis of how easily these tools can be deployed and used redily.
2. Ealuation for level of detail of measurements- In this category, we evaluated the functionality of the tools i.e. what all interactions the tool is capable of capturing.
3. Additional system information - In this category, we provided the additional system information that might be needed by users to choose one of these tools according to their requirement.

The following criteria are used to establish comparison between these tools:

Evaluation for ease of deployment:

1) Easy to install
2) Time required to install
3) User interface
4) Tutorials available
5) Open source
6) Programming skills required
7) Documentation provided
8) Examples provided
9) User interface for developers

Evalution for level of detail of measurements:

1) Interaction with text fields

2) Mouse movements
3) Mouse clicks
4) Mouse drags
5) Scrolling
6) Dialog box

Additional system information:

1) Size
2) Platforms
3) Browser support
4) Additional features

V. Comparison Analysis

All of the tools we tested are at least functional: they capture interactive web sessions and successfully play them back at a later time. While taking all criteria into consideration, Sahi performs better overall than the other tools we tested.

Compared to Telerik and Reanimator, Sahi has easier installation. It has strong documentation available online. It does not require any programming skill to use Sahi. Sahi supports Chrome, Firefox and Internet Explorer. Selenium is easiest to install as it comes as firefox plugin. It also provides all the features like Sahi but the greatest setback of selenium is that it can only be used to capture and replay Firefox sessions. It has a great feature that allows the user to set speed to analyse the scenarios. No other tool provides this feature and has their own replay speed. This makes the process of analysis cumbersome if the default speed is very fast or slow. Telerik despite being a great tool has some major issues.The biggest setback of Telerik is that its installation takes very long (it took us 1 day to install) to finish since it requires lot of dependencies and also its installation process is not very effective. It requires large hard disk space. It has the feature called visual storyboard which gives the snapshot of each scenario recorded by tool. This allows the developer to go to any snapshot to analyse the interaction. We used the one month free trial version of Telerik and requires to purchase once the trial period expires. Telerik also supports Chrome, Firefox and Internet Explorer. Reanimator is a very basic capture and replay tool inspired by Mugshot. It requires programming skills to get it working since it does not come as ready to use testing tool. It is built on JQuery 1.8.3. Also it works only on Firefox and Internet Explorer. It does not have any additional feauture like the other tools.

Later, we talk about modifications we made to Reanimator that made it more functional and easier to use. We compared the four tools, however, as we found them directly from their authors. We did not compare our upgraded version of Reanimator during this test.

Table 1 shows the comparison between Reanimator, Sahi, Telerik and Selenium based on criteria for evalution for ease of deployment. Table 2 shows the comparison based on criteria for evalution for level of detail of measurements. Table 3 shows the additional system information provided with each tool. The symbol ✓ means that the tool provides the characteristic which is in evaluation and the symbol ✗ means that it doesnt support it.
Based on the evalution criteria seen in table 1, 2 and 3, we choose Sahi as the best tool among all others.

VI. Extension and Modification of Reanimator

We forked the original Reanimator GitHub project at https://github.com/brianmpollack/reanimator. It took awhile to determine exactly how the original authors built Reanimator, but we were able to understand their work to determine how Reanimator was built.

A. Original Reanimator Code

Reanimator is built as an extension to JQuery and uses NodeJS to compile the source files. The compile scripts start with JQuery and use Almond to inject the Reanimator codebase. The main driver of Reanimator is contained in the lib/reanimator/core.js file. There is a file in the lib/reanimator/plugins directory which serves as the implementation for each logging feature that Reanimator supports.

B. Issues with Original Code

The original Reanimator code contained a few runtime and compilation issues, which we fixed before continuing to work with Reanimator.

Characteristic	Reanimator	Sahi	Telerik	Selenium IDE
Easy to Install	✗	✓	✓	✓
Time Required to Install	Very Quick	Quick	Long and Tedious	Quick
User Interface	✗	✓	✓	✓
Easy to Use	✗	✓	✓	✓
Documentaion	✗	✓	✓	✓
Tutorial	✗	✗	✓	✓
Open Source	✓	✓	✗	✓
Examples	✓	✓	✓	✓
Programming Skills	✓	✗	✗	✗

TABLE I: Evaluation for ease of deployment

Characteristic	Reanimator	Sahi	Telerik	Selenium IDE
Mouse Click	✓	✓	✓	✓
Text Field	✗	✓	✓	✓
Mouse Drags	✗	✗	✗	✗
Mouse Move	✗	✗	✗	✗
Scrolling	✗	✗	✗	✗
Dialog Box	✗	✓	✓	✓

TABLE II: Evaluation for detail of measurement

Characteristic	Reanimator	Sahi	Telerik	Selenium IDE
Size	2.51MB	25.2MB	194MB	16KB
Platforms	Windows, Linux	Windws	Windows	Windows Linux
Browser Supporrt	Firefox	Chrome, Firefox, IE	Chrome, Firefox, IE	Firefox
Additional feature	None	None	Storyboard	Set speed

TABLE III: Additional system information

1) Google Chrome: First, we observed that the precompiled code in the demos did not work in Google Chrome. It appears that the original authors tried to convert DOM objects to JSON. DOM stands for Document Object Model and JavaScript uses DOM Objects to represent the webpage contents.

It is commonly known that JSON does not support circular references, but DOM Objects contain numerous circular references. Some browsers, such as Firefox, will simply proceed with the JSON encoding and will automatically drop the circular references, but other browsers, such as Chrome, will terminate the JavaScript process. We found a simple function that will remove the circular references from the DOM object while converting it to JSON, allowing Reanimator to now be used in any major browser (this fixed Reanimator in our tests with Chrome, Firefox, Safari, Internet Explorer, and

Edge).

2) File Link Error: We also found a file error in the included demos. The original Makefile used a symbolic link to link the Reanimator JavaScript code to the demos, rather than copying the files. The symbolic link, however, caused issues both when the code is uploaded to GitHub and even when running locally.

We updated the Makefile to copy the files into the demo folders. While we could have updated the demo code to link to the dist directory of the Reanimator code, we felt that it was important to allow the demos to be self-contained. This would allow a user to copy the contents of a demo folder directly to their server for testing.

C. Uploading Log to Server

Originally, Reanimator was not very useful to developers in the field. As soon as a user closed his or her browser, any recorded log would be destroyed. To make Reanimator more useful to developers, the main feature we wanted to add was log uploading.

1) Client Side: We refer to the Reanimator JavaScript code as the client side because it is running locally on the web clients machine. To enable log uploading, we modified the capture function in the core.js file. We added parameters so the user could pass in a boolean to enable uploading, and so the client could pass in a UUID to be associated with the log. A UUID, or Universally Unique Identifier, is a unique ID for the client. Generation and use of UUIDs is outlined in RFC 4122. [6]

The client, if log uploading is enabled, will upload the log to an HTTP POST service every five seconds. By default, we programmed Reanimator to POST to a file called reanimator.php in the same directory as the JavaScript file.

We duplicated and extended the original tile game demo to support uploading to a webservice. A user can play the tile game, then can select a list of sessions from a dropdown. After selection, he or she can see any past game that was logged to the server.

2) Server Side: The server can be written any way the developer prefers as long as it accepts HTTP POST requests.

We included a sample HTTP POST web service file in the dist directory of Reanimator. The simple script is written in PHP and will upload the logs to a MySQL database for retrieval later. It will overwrite any old logs with the same UUID.

We also included a sample database configuration file with the uploading tile game demo. Because a developer should know how to setup a simple web service and MySQL server, we do not feel the need to explain them in detail.

VII. Future Additions to Reanimator

In the future, we recommend adding more features to Reanimator. We believe that it is useful to developers after our additions, but could be improved further.

A. Developer Interface on Playback

We propose creating a developer interface, or developer portal. It would provide a user friendly way for developers to organize and replay their captured sessions. The interface could include a play/pause button, and let the developer choose the speed to play back the session.

Because this does not add any core functionality, but rather an improved user experience, we recognize that it would help developers but we did not choose to implement it during our Reanimator extension/proof of usefulness test.

B. Synchronize Only Changes

Our demonstration implementation will upload the user's log every five seconds. The server code will simply overwrite the log data for that user in the database when it receives new data. This practice is fine for demo purposes and for small websites, but is obviously not scalable to large websites. We propose uploading only the changes of the log to the server.

To ensure that data is not lost, we could allow the server to respond to the POST request to tell the client if the log upload was successful. The client would keep track of successful log pushes and would know if it needs to retry or if it can upload a new section of the log.

This modification would appear small to each client, but would be a huge difference to large websites which could potentially be receiving many log pushes at the same time.

C. Smaller Upload Size

The log files that Reanimator generates are large relative to the usefull information that they contain. The files contain every piece of information about the layout of the webpage. We propose modifying the way that logs are stored. There will be one initial log with the entire DOM, but subsequent events will simply be a changelog of the original log file.

This would make the entire log smaller, allowing for smaller uploads and less memory used at the client.

It is important to note that this solution is different from the previous suggestion (synchronizing only the changes). The entire log would be much smaller, as it would only contain the changes to the original log file. We can combine this with the previous suggestion to further minimize the overhead of Reanimator.

VIII. Conclusion

A. Tool Comparison

In our study, we compared four capture and replay tools: Reanimator, Sahi, Telerik, and Selenium. The goal of our study was to determine the best tool among these four for a developer to use in the field. We compared the four tools on different criteria to find out which tool has reliable capture and replay capabilities without requiring too much time to set up and use.

After evaluating all four tools, we came to the conclusion that Sahi best satisfies our requirements. We believe that Sahi has the best overall performance and can be easily adapted by any developer to test their GUI application.

B. Reanimator Extension

We modified Reanimator with the expectation that it would be more useful for JavaScript developers to deploy to their applications in the field. We fixed bugs present in the original code to make Reanimator compatable with Google Chrome, upgraded it to support JQuery 3.2.0, and extended the functionality to allow developers to request the logs be uploaded to a server.

Because the original code for Reanimator has not been modified in four years, we assumed that the developers abandoned the project. Because our version was siginificantly different from the original, we decided to not issue a pull request back to the original codebase.

We believe that the original version of Reanimator was novel, but not very useful because the log was lost when a user exited his or her browser. With the addition of persistent logs (if they are being uploaded to a server), we believe that Reanimator is a very solid, lightweight capture and replay tool. We recommend that JavaScript developers implement it if they need to capture logs.

We recommend adding a few more featurers to Reanimator. We recommend making the log filesizes smaller and adding a user interface for developers to help them navigate through a user's session.

IX. Summary of Changes

A. Tool Comparison

Before our initial progress report, we had already made great progress with our analysis of different capture and replay tools. We wanted to perform the analysis early so that we could focus on the extension to Reanimator, which we viewed as the more important part of this project, since it involved creating new features that would be useful to developers.

Since our initial progress report, we expanded our study of previous work done in the field of capture and replay by studying "Time Travel Debugging for Javascript/Node.js". This is one of the more recently developed tools in the field that allows a developer to simply press a button to step-back in a program's execution without requiring the developer to manually stop debugging, set a new breakpoint, and then reproduce the desired execution to hit that breakpoint. We were impressed with this tool.

B. Reanimator Extension

At the time of our initial progress report, we had upgraded Reanimator to work with JQuery 3.2.0. We also fixed the bugs in the code which prevented Google Chrome users from recording a session. We had to spend a significant amount of time learning how the original authors compiled the original version of Reanimator, since they left out a few important dependencies in the package.json dependency list.

Since our initial progress report, we have implemented and tested the upload feature of Reanimator. We also built a new demo page to showcase the new feature.

REFERENCES

[1] Earl T. Barr and Mark Marron. "Tardis: Affordable Time-travel Debugging in Managed Runtimes". In: *SIGPLAN Not.* 49.10 (Oct. 2014), pp. 67–82. ISSN: 0362-1340. DOI: 10.1145/2714064.2660209. URL: http://doi.acm.org/10.1145/2714064.2660209.

[2] Earl T. Barr et al. "Time-Travel Debugging for JavaScript/Node.js". In: Association for Computing Machinery, Sept. 2016. URL: https://www.microsoft.com/en-us/research/publication/time-travel-debugging-javascriptnode-js/.

[3] Scott McMaster and Atif M. Memon. "An Extensible Heuristic-Based Framework for GUI Test Case Maintenance". In: *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops*. ICSTW '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 251–254. ISBN: 978-0-7695-3671-2. DOI: 10.1109/ICSTW.2009.11. URL: http://dx.doi.org/10.1109/ICSTW.2009.11.

[4] James Mickens, Jeremy Elson, and Jon Howell. "Mugshot: Deterministic Capture and Replay for Javascript Applications". In: *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*. NSDI'10. San Jose, California: USENIX Association, 2010, pp. 11–11. URL: http://dl.acm.org/citation.cfm?id=1855711.1855722.

[5] Stanislava Nedyalkova and Jorge Bernardino. "Open Source Capture and Replay Tools Comparison". In: *Proceedings of the International C* Conference on Computer Science and Software Engineering*. C3S2E '13. Porto, Portugal: ACM, 2013, pp. 117–119. ISBN: 978-1-4503-1976-8. DOI: 10.1145/2494444.2494464. URL: http://doi.acm.org/10.1145/2494444.2494464.

[6] R. Salz P. Leach M. Mealling. *A Universally Unique IDentifier (UUID) URN Namespace*. RFC 4122. RFC Editor, July 2005.

[7] Elder M. Rodrigues et al. "Evaluating Capture and Replay and Model-based Performance Testing Tools: An Empirical Comparison". In: *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM '14. Torino, Italy: ACM, 2014, 9:1–9:8. ISBN: 978-1-4503-2774-9. DOI: 10.1145/2652524.2652587. URL: http://doi.acm.org/10.1145/2652524.2652587.

[8] Michael Silverstein. "Logical Capture/Replay". In: *STQE Magazine* 5.6 (2003), pp. 36–42.