Continuous Integration for Web-Based Software Infrastructures: Lessons Learned on the *webinos* Project

Tao Su[1], John Lyle[2], Andrea Atzeni[1], Shamal Faily[3], Habib Virji[4], Christos Ntanos[5], and Christos Botsikas[5]

[1] Dip. di Automatica e Informatica, Politecnico di Torino, 10129 Torino, Italy
{tao.su,shocked}@polito.it
[2] Department of Computer Science, University of Oxford, UK
johnplyle@gmail.com
[3] School of Design, Engineering & Computing, Bournemouth University, UK
sfaily@bournemouth.ac.uk
[4] Samsung Electronics UK
habib.virji@samsung.com
[5] National Technical University of Athens
{cntanos,cbot}@epu.ntua.gr

Abstract. Testing web-based software infrastructures is challenging. The need to interact with different services running on different devices, with different expectations for security and privacy contributes not only to the complexity of the infrastructure, but also to the approaches necessary to test it. Moreover, as large-scale systems, such infrastructures may be developed by distributed teams simultaneously making changes to APIs and critical components that implement them. In this paper, we describe our experiences testing one such infrastructure – the *webinos* software platform – and the lessons learned tackling the challenges faced. While ultimately these challenges were impossible to overcome, this paper explores the techniques that worked most effectively and makes recommendations for developers and teams in similar situations. In particular, our experiences with continuous integration and automated testing processes are described and analysed.

Keywords: continuous integration, automated testing, web-based software infrastructure, functional testing.

1 Introduction

As web-apps become more pervasive, the reliance on web-based software infrastructures, such as middleware products and libraries, is growing. However, because end-users do not interact directly with such infrastructures, then devising appropriate strategies for detecting and preventing hidden software defects is challenging.

Four complementary approaches are typically used to identify such defects. *Unit tests* target isolated modules, *integration tests* target the integration of

V. Bertacco and A. Legay (Eds.): HVC 2013, LNCS 8244, pp. 145–150, 2013.

components, *functionality tests* target source code functions, and *system tests* target high level functionality. Carrying out these tests sequentially can be time-consuming as developers need to wait for the results of these tests before they can continue working. Moreover, integration problems of the source code of different contributors commonly occur for a myriad of reasons. To address these problems, Fowler [1] introduced the idea of *continuous integration*. This entails continuously downloading, integrating, and testing the source code and project libraries contributed by each developer. Following this approach, software can be tested for defects that might not otherwise be noticed by an individual developer.

A continuous integration strategy is difficult to devise when testing distributed software infrastructures because it needs to simulate all required devices and services. In this paper, we describe our experiences developing and applying continuous integration to test the *webinos* platform. *webinos* provides an overlay network between an individual user's set of personal devices, including their PC, smartphone and TV, and then allows web applications to access services on these devices through a set of standard JavaScript APIs. The complex interactions between web-based devices through browsers make *webinos* particularly hard to effectively test from one side, and particularly interesting as a case study from the other.

2 Related Work

Most web-based application testing researches focus on client-server applications that implement a strictly serialised model of interactions, generating test cases based on user-session profiling [2][3], or on testing the correct functioning on the client side or the server side separately [4][5].

Currently, there are no similar methods to test web-based software infrastructures like *webinos*, which appears to be the first platform for sharing services from different types of devices through browsers. One approach for using continuous integration testing is described in [6]. The authors presented a neat plug-in for Selenium implemented on a continuous integration server; this hooks every AJAX call made by the tested web application to verify requested data before application processing. This helps narrow down the location of a fault irrespective of whether it is situated in server- or client-side code. However, compared with our approach, this work only narrows the error position, and it is already covered by available functionality tests.

3 *webinos* Architecture

webinos is a secure platform which can be accessed by multiple types of web-enabled devices. In its life cycle, more than 30 organizations, mostly based in Europe, contributed to its development. It introduces the *Personal Zone* concept, where all the devices (*Personal Zone Proxies*) belonging to the same zone support and expose standard JavaScript APIs for accessing services such as device features (cameras, geolocation, networking and etc).

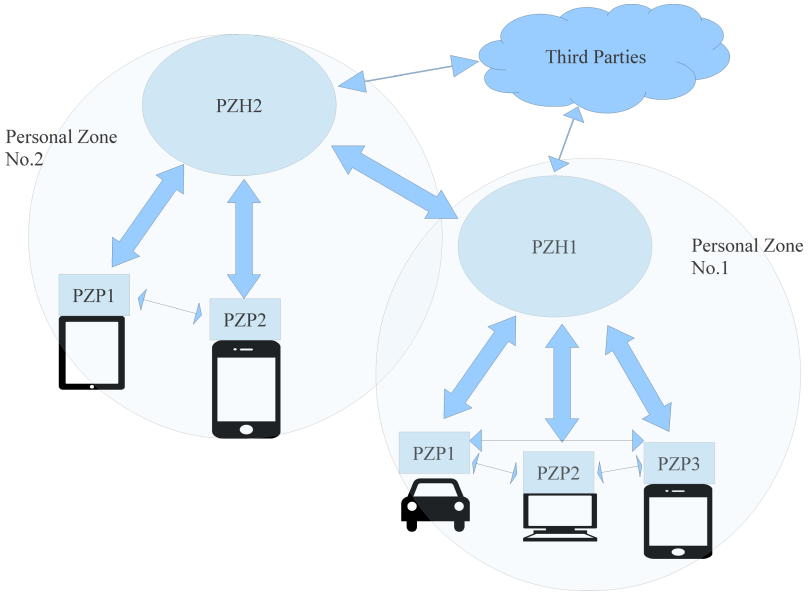

Fig. 1. *webinos* architecture

As shown in Figure 1, the *Personal Zone Hub* (PZH) is the focal point of the personal zone. The interactions might take place between PZHs and PZPs in the same zone or from different zones in different modes, when they try to share the resources or communicate with each others.

4 Testing *webinos*

As a web-based software infrastructure, *webinos* is designed to work on different browser-enabled systems and devices. This makes the testing extremely challenging. The interpretations of different browsers for the same JavaScript code are not identical, especially for browsers with different JavaScript engines. Implementing the same testing code on device with different architectures and computing power, is also challenging. For example, modules accessing OS specific functionality such as accessing path or network connectivity, provide different responses.

To simplify the development process, the platform was modularised; this meant that each API had its own git repository. This also made testing more complex. Since the APIs and core components are under continuously development by groups which apply rapid development methodology, it would be very easy for certain API to break other components and even the whole platform despite having passed its own unit test.

The testing infrastructure was created relatively late in the project. This meant that testing engineers were busy catching up with the developed compo-

nents, and needed to generate multiple levels of tests to ensure their components worked correctly independently and when integrated with *webinos*.

4.1 Approach

To overcome the challenges listed above, we introduced the continuous integration technique, which means downloading source code, integrating and running various levels of tests continuously. Five levels of tests are performed, the API unit tests, integration tests, functionality tests, system tests and APIs' integrate-with-*webinos* tests.

The API unit tests are executed first to test only the module and its dependencies. After that, the integration tests, functionality tests and the system tests are executed to thoroughly test the integrated platform. The functions in the source code are called directly in functionality tests, and the results are generated by comparing the returned with the predefined values. This step requires the help of a JavaScript test framework. The system tests are executed in a higher level, mimicking user interactions. Thus, to simulate real case operations, a headless browser is used to automate this step. After the unit tests and platform tests, the APIs are integrated with the *webinos* platform one after another, testing if they can work correctly after integration.

Setting the order as mentioned is to make the discovery of errors easier: each unit test is delimited to a specific module, so the test failure means the problem is circumscribed to the module or its dependencies. The integration tests and functionality tests are executed before the system tests for similar reason, since they are more comprehensive and can provide more information on why the test fails.

To minimise the testing work, the tests are integrated with continuous integration servers. They are set to download the newest source code, execute the listed tests one after another in the defined order, and notify the testing engineers if error happens. For unit tests which only take a short time, the continuous integration server is set to perform the tests after the developer commits, while the other tests which take longer time are performed in the middle of the night.

4.2 Lessons Learned

The sub-sections below characterise three lessons learned in developing and applying our continuous integration system for *webinos*.

Continuous Integration Is Shaped by the Revision Control System Used. In *webinos* project, git is chosen to revision control source code. In the github model, developers build components within their own sandboxes and once the components are stable, they ask for pull request to the official repository running on github. Automated tests of pull requests minimises the time that maintainers spend reviewing submissions, but delays accepting pull requests cause merge conflicts, means that multiple incompatible pull requests occur. Therefore, open source development approaches using systems like github need very

active maintainers and prioritise accepting contribution. For the maintainer who is also a developer, the rule of "you should not merge your own pull request" is effective to avoid errors brought by blind confidence, but causes the mentioned problem.

As an example, several thumb developers worked together and updated the *webinos-pzh* module, but according to the rule, they can not merge their pull request themselves. Therefore discussions were held on this pull request. However, at the same time the other pull requests were merged directly. After the discussion, the pending request was merged, caused a lot of failed tests, these developers had to rework the updates to incorporate with the new merged changes.

Maintaining the Test Infrastructure Is Harder Than Maintaining the System. For *webinos*, a comprehensive testing system may be more difficult to create than the infrastructure itself.

As an example, to test *webinos* on various OSes, we use three different continuous integration servers: one cloud-based infrastructure, one self-hosted infrastructure and one self-developed node.js module. The infrastructures focus on Linux platforms, which do not work very well on proprietary platforms. Therefore the node.js module is developed to cover this inadequacy. Beside that, the tests also break frequently because of rapid development of the components.

From the *webinos* experience, we believe the best option should start testing from low level to up level as state in section 4.1. The test cases for individual components can be generated by the developers, this way may fasten the testing procedure. Also in the most cohesive and loosely coupled components, fairly good unit tests would help to discover some problems which should be found in the integration tests. Integration and functional tests work better if it is possible to assign several developers who have detailed knowledge of the source code to write these test cases, as these tests are the most important part for testing a modularised system like *webinos*. Even if this would require a quite large amount of resources, bugs raised in the cooperation with other modules are exactly the kind of problem that an individual developer can not find out. Generating the system test cases is easier, the developers of these test cases only need to know how to operate with the *webinos* platform and use a headless browser to simulate the operations.

Thus, in our opinion, the best way to assign the resources and speed up the testing procedure is to assign developers the individual component test responsibility. Also system tests can be generated by single developers, while more care and resources are needed for integration and functional tests, requiring cooperation from different modules developers.

Developers Only Test for a Single Platform. Developing *webinos* for multiple platforms raised several issues that were hard to overcome. Ideally, the dependencies and modules should be tested on all the platforms before further implementation and development. For the reason that dependencies or modules may behave differently on each platform at runtime. Similarly, the binaries for

native modules should be compiled separately on all platforms. In reality most developers worked only on a single platform at a time, and tend to only concern the tests passed on their own systems. This may led to subsequent bugs and incompatibilities piling up on other platforms.

For example, *webinos* widget packaging required the *zipfile* library; which was included and built on Linux developer's machines. Although fully functional at runtime on both Linux and Windows machines, the library failed to be built on Windows. Even though the testing system was totally functioning, this error remained undetected for several months until a Windows developer tried to compile it. This subsequently led to several days being spent re-adapting this library.

5 Conclusion and Future Work

Using continuous integration system to automate various levels tests increases development efficiency, it also increases our confidence about the quality of *webinos* platform. However, the testing system has its own limitations. At present, the system is unable to cover all of *webinos*' supported operating systems. Since *webinos* applies security as a pre-requisite, security tests are being introduced as an important part of our testing approach in the future.

Acknowledgements. The research described in this paper was funded by the EU FP7 *webinos* project (FP7-ICT-2009-05 Objective 1.2).

References

1. Fowle, M.: Continuous integration in martin fowler's blog (2000), `http://martinfowler.com/articles/continuousIntegration.html`
2. Sampath, S., Sprenkle, S., Gibson, E., Pollock, L., Greenwald, A.S.: Applying concept analysis to user-session-based testing of web applications. IEEE Transactions on Software Engineering 33(10), 643–658 (2007)
3. Elbaum, S., Rothermel, G., Karre, S., Fisher, M.: Leveraging user-session data to support web application testing. IEEE Transactions on Software Engineering 31(3), 187–202 (2005)
4. Di Lucca, G.: Testing web-based applications: the state of the art and future trends. In: 29th Annual International Computer Software and Applications Conference, COMPSAC 2005, vol. 2, pp. 65–69 (2005)
5. Marin, B., Vos, T., Giachetti, G., Baars, A., Tonella, P.: Towards testing future web applications. In: 2011 Fifth International Conference on Research Challenges in Information Science (RCIS), pp. 1–12 (2011)
6. Falah, B., Hasri, M., Schwaiger, S.: Continuous integration testing of web applications by sanitizing program input. Cyber Journals: Multidisciplinary Journals in Science and Technology 3(2) (2013)