LMU Munich
Institute of Informatics
Teaching and Research Unit Programming and Modelling Languages

Bachelor Thesis

Leveraging Human Computation for Quality Assurance in Open Source Communities

J. Sprinz

Principal Advisor Prof. Dr. François Bry
Date of submission 29.01.2022

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Zitate wurden als solche kenntlich gemacht.

München, den 29.01.2022

Sprinz

J. Sprinz

”We all want to help one another. Human beings are like that. We want to live by each other’s happiness - not by each other’s misery. We don’t want to hate and despise one another. In this world there is room for everyone. And the good earth is rich and can provide for everyone. The way of life can be free and beautiful, but we have lost the way.

Greed has poisoned men’s souls. Has barricaded the world with hate. Has goose-stepped us into misery and bloodshed. We have developed speed, but we have shut ourselves in. Machinery that gives abundance has left us in want. Our knowledge has made us cynical. Our cleverness, hard and unkind. We think too much and feel too little. More than machinery we need humanity. More than cleverness we need kindness and gentleness. Without these qualities, life will be violent and all will be lost.

The aeroplane and the radio have brought us closer together. The very nature of these inventions cries out for the goodness in men - cries out for universal brotherhood - for the unity of us all. Even now my voice is reaching millions throughout the world - millions of despairing men, women, and little children - victims of a system that makes men torture and imprison innocent people.

To those who can hear me, I say - do not despair. The misery that is now upon us is but the passing of greed - the bitterness of men who fear the way of human progress. The hate of men will pass, and dictators die, and the power they took from the people will return to the people.”

- Charlie Chaplin in *The Great Dictator* (1940)

Acknowledgements

First, I want to express my gratitude to the fantastic people at the Teaching and Research Unit Programming and Modelling Languages at LMU Munich. Thank you to Prof. Dr. François Bry for giving me the opportunity to work at the department and for your support in finalizing this thesis. I want to extend special gratitude towards Dres. Sebastian Mader and Niels Heller, who mentored me in the early stages of this project and provided valuable hints that both motivated and challenged me.

Thank you for everyone's enthusiastic participation in the user study - it was absolutely marvelous! My biggest thanks to the Ubuntu Community for allowing me to be a part of countless exceptional projects over the years. These past years of working on Ubuntu Touch has been the most rewarding experience of my life. Time and time again, it is simply wonderful to see a community driven by the desire to make a positive difference and accomplish greatness.

Finally, I want to thank my amazing partner for their emotional support while completing this thesis. You mean the world to me!

Converging Interest Disclosure

In accordance with my ethical obligation as a researcher, I am reporting that I am an active member of the Board of Directors of the UBports Foundation, a non-profit entity that has inspired and could benefit from the work reported in this thesis.

Zusammenfassung

Software, die unter dem quelloffenen Entwicklungsmodell (Open Source Development Model, OSSD) entwickelt wird, hat in den letzten Jahrzehnten massiv an Bedeutung gewonnen. Immer mehr kritische Komponenten werden im Rahmen des OSSD entwickelt und der Bedarf für tiefgehende Qualitätssicherung (Quality Assurance, QA) steigt. Diese Arbeit untersucht, ob das OSSD Potenzial für formalisierte Nutzertests durch unerfahrene freiwillige Community-Mitglieder bietet. Eine Human Computation Plattform zur Sammlung solcher Testergebnisse wurde entworfen und trägt den Namen open crowdsourced user-testing suite (OPEN-CUTS). Eine mit einem Prototyp von OPEN-CUTS durchgeführte Benutzbarkeitsstudie bestätigt diesen Ansatz und zeigt mögliche zukünftige Forschungsfragen auf.

Abstract

Software developed under the open source development model (OSSD) has risen to significant importance over the recent decades. With more and more critical components being developed under the OSSD, the need for extensive quality assurance (QA) increases. This thesis investigates any potential for conducting formalized user testing through inexperienced volunteer community members under the OSSD. A human computation platform to aggregate such test results was designed and named open crowdsourced user-testing suite (OPEN-CUTS). A usability study of a prototype of OPEN-CUTS confirms the viability of this approach and points to potential future research questions.

Contents

List of Figures

Chapter 1

Introduction

Software developed within the open source development model has risen to significant importance. WordPress, Blender, and Linux are only a few examples of free and open source software projects that have revolutionized the software market over recent decades. Besides being distributed free of charge and their source code being publicly available, what sets open source software apart are the volunteer communities that support or even drive their development process. This property creates particular challenges and opportunities related to software quality assurance, which will be explored in this thesis. Software quality assurance, meaning ensuring a software product's fitness for use, plays a significant role in modern software development practices. This thesis investigates whether the presence of volunteers in the development process creates a potential for crowdsourced user testing. A software solution is proposed to facilitate formalized user testing in open source communities through crowdsourcing and human computation. The requirements for such a solution are drawn from examining the needs of an exemplary open source community. Finally, a prototype for this software solution is implemented and evaluated through a usability study.

1.1 Objective

This project aims to identify and address challenges open source communities face when doing quality assurance. The goal is to develop a prototype of a human computation platform that facilitates the quality assurance process by having inexperienced community members generate formalized user testing data. The software should give users concise instructions for conducting tests and recording their results in an easy-to-use interface. Test results from all testers should then be aggregated and presented to the developers in a meaningful way. If this prototype's evaluation shows evidence for its usefulness, it should be possible to compile a list of future development goals and prepare for a long-term study.

1.2 Contribution

A human computation platform for the previously described purpose has been designed and implemented. An early software prototype was evaluated in a usability study, which yielded promising results and generated valuable insights into potential future de-

velopment goals. The software has been named OPEN-CUTS, the open crowdsourced user testing suite. To enable its continued development, OPEN-CUTS has been made available to the public as an open source project: https://www.open-cuts.org.

1.3 Overview

Chapter 2 outlines the theoretical principles that this thesis builds on and provides an overview of the literature. Next, chapter 3 provides background information on the work done in this thesis. The UBports Community is introduced as a reference community with different use cases for formalized user testing. Two existing software solutions are analyzed to illustrate the need for a new solution. Building on this, a blueprint for a software solution is outlined. Chapter 4 discusses the implementation of this blueprint with particular attention to technical details. The resulting prototype is evaluated in chapter 5. A usability study with members of the UBports Community is designed, conducted, and analyzed. The thesis is concluded in chapter 6, which summarizes the findings and outlines future work.

Chapter 2

Related Work

This chapter outlines the theoretical principles that this thesis builds on, establishes definitions of essential terms, and provides an overview of the literature.

2.1 Human Computation, Crowdsourcing, Gamification, Games with a Purpose

Information technology strives to enable full automation of "any operations which could be done by a human" [1, p. 437]. The exponential increase in processing power[1] combined with constant advances in traditional algorithms and new technologies such as machine learning[2] have made this goal seem more and more attainable. However, many tasks that are trivial to a human are still impossible to automate for machine execution. The practice of outsourcing such computation steps from a machine to a human agent is called *human computation* (HC) [4], [5].

A related concept is *crowdsourcing* (CS), meaning the involvement of many people in achieving a common goal. The internet has enabled a new collaboration culture that allows large-scale independent parallel work by removing physical barriers and drastically reducing coordination overhead [6, p. 89]. Well-known examples of successful CS platforms include collaborative projects like Wikipedia[3], but the term also applies to platforms that facilitate HC through a large group of human agents. Platforms that combine HC and CS typically automate coordination by continuously assigning agents small tasks. One such platform is Mozilla Common Voice[4], which aims to create a corpus of transcribed speech in the public domain. "To achieve scale and sustainability, the Common Voice project employs crowdsourcing for both data collection and data validation." [7, p. 1]. Amazon's Mechanical Turk[5] (MTurk) is an example of a commercial CS platform that allows "[posting] tasks to workers to perform in return for monetary payment" [4, p. 46].

[1] An effect known as *moore's law* [2].

[2] Machine learning refers to programs that improve automatically through training, for example, by identifying statistical correlations in large sets of data [3].

[3] A public-domain encyclopedia that can be edited by everyone, see https://www.wikipedia.org/.

[4] https://voice.mozilla.org/

[5] https://www.mturk.com/

Human agents need some kind of incentive to participate in HC or CS software. Besides motivational factors such as providing monetary compensation (MTurk) or offering the experience of contributing to a good cause (Wikipedia, Mozilla Common Voice), users can be motivated by making the contribution process more fun and engaging using game-like mechanisms. Mozilla Common Voice, for example, allows participants to define weekly activity goals. Users are then motivated to achieve these goals in various parts of the software. This practice is known as *gamification* [8].

So-called *games with a purpose* (GWAP) further extend gamification to the point where the agents do not even need to be aware that their actions contribute to HC because the data is a mere by-product of the game [4, p. 63]. The concept was pioneered by von Ahn and Dabbish to create a game to label images on the web [9]. A well-known example of a GWAP is ReCaptcha, a mechanism to introduce additional complexity to forms on the world wide web to prevent automated access [4, p. 48]. The software forces website visitors to complete a simple task such as transcribing a scanned word to train optical character recognition software [4, p. 49]. HC also includes some *data-mining* techniques, such as the practice of assessing the relevance of web search results based on user actions [10, p. 86], [5, p. 8].

2.2 Quality Assurance in Software Engineering, Software Testing

Quality Assurance (QA) in software engineering covers all activities throughout the development process that ensure a software product's "fitness for use" [11, p. 7]. Besides design-related and organizational activities, testing plays a significant role in the QA process. Tests can be classified based on their observed level of detail [11, p. 257]:

- *Unit testing* focuses on the functionality of atomic software components, such as a single function call. Dummy procedures (a practice called *mocking* or *stubbing*) replace all external functionality to ensure that the test result only reflects the unit under test.
- *Integration testing* examines integrated modules of the software to ensure interoperability. Multiple levels of integration can be iterated by gradually including more components.
- *System testing* observes the complete software product.

Tests can be automatically executed by a computer (*automated testing*) or run manually by a human agent (*user testing*). The latter is typically only used for system testing since these tests are often difficult to automate and relatively easy to perform by humans. Since every tester will approach the system with their individual perspective, user testing might also result in constructive feedback beyond the test's original scope - an unattainable benefit for automated testing. User testing is called *formalized user testing* if it follows a plan consisting of specific instructions or *smoke testing* if the tests cover all system functionality.

2.3 Open Source Software Development Model

Open source software (OSS) refers to "a software product with the source code made public so that anyone can read, analyze, and change or improve the code" [12, p. 459]. OSS is typically developed in online volunteer communities. The sustainability of these communities is vital for the software product's success. Aberdour's Onion Model (see Figure 2.1) depicts the structure commonly seen in sustainable OSS communities: The community consists of "a small number of core developers and increasing numbers of contributing developers, bug reporters, and users. 'Onion' refers to the successive layers of member types" [13, p. 59]. Community members may choose to increase their involvement on their own terms, thus progressing towards the core.

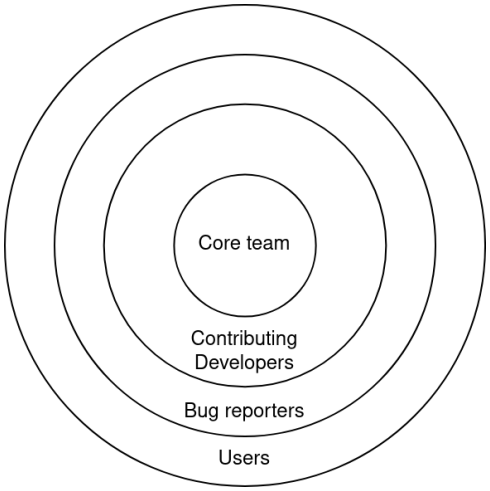


Figure 2.1: The onion model. Graphic based on [13, p. 59].

The open source software development model (OSSD) has the distinct property of publicly available source code. Raymond observes that more people repeatedly looking at the source code and reviewing its changes results in higher quality code [14]. Additionally, a volunteer-based team motivated by different factors can multitask more efficiently, e.g., by independently working on different issues important to different parts of the community and designing, developing, testing, and debugging the software in parallel, thus enabling rapid progression in small increments.

2.4 Quality Assurance under the Open Source Software Development Model

OSS communities find themselves in the fortunate situation of having a large number of volunteers at their disposal who are willing to "take ownership of system testing and bug reporting. [...] The sheer size of the bug-reporting group will ensure that more people test the system (on more platforms) than any commercial organization could hope to achieve" [13, p. 60].

Wahyudin et al. have developed a framework to classify QA activities under the OSSD, depicted in Figure 2.2. Three groups of activities are distinguished [15]:

I **Defect detection**

1. A defect is identified and discussed on a community-support platform such as a chatroom, forum, or mailing list. Defect detection can occur unplanned during everyday use or planned in the context of user testing.
2. If the defect does not generate sufficient interest in the community, it might just be ignored (a). Otherwise, it may be or reported to the bug tracker by another more experienced user (b), or directly by the user who first identified the defect (c).

II **Defect verification**

3. *Defect collection*: Further analysis of the defect to verify its existence in a particular software release. More information about the defect is collected, and it may be categorized regarding severity and affected software features.
4. *Defect fixing*: An interested developer will take necessary steps for mitigation and possibly communicate a short-term plan with the community.

III **Solution verification**

5. *Code self-review*: The developer verifies their proposed change locally.
6. *Peer-review*: Other members of the community review the proposed change. Contributing developers will try to understand the proposed change's technical details, and bug reporters will test the software to ensure that the defect has been resolved and no new defects have been introduced.

Automated testing is not represented in this framework. Since the original publishing of this framework in 2007, the availability of automated testing tools has increased. Today, automated testing often plays a significant role in solution verification [16, p. 1].

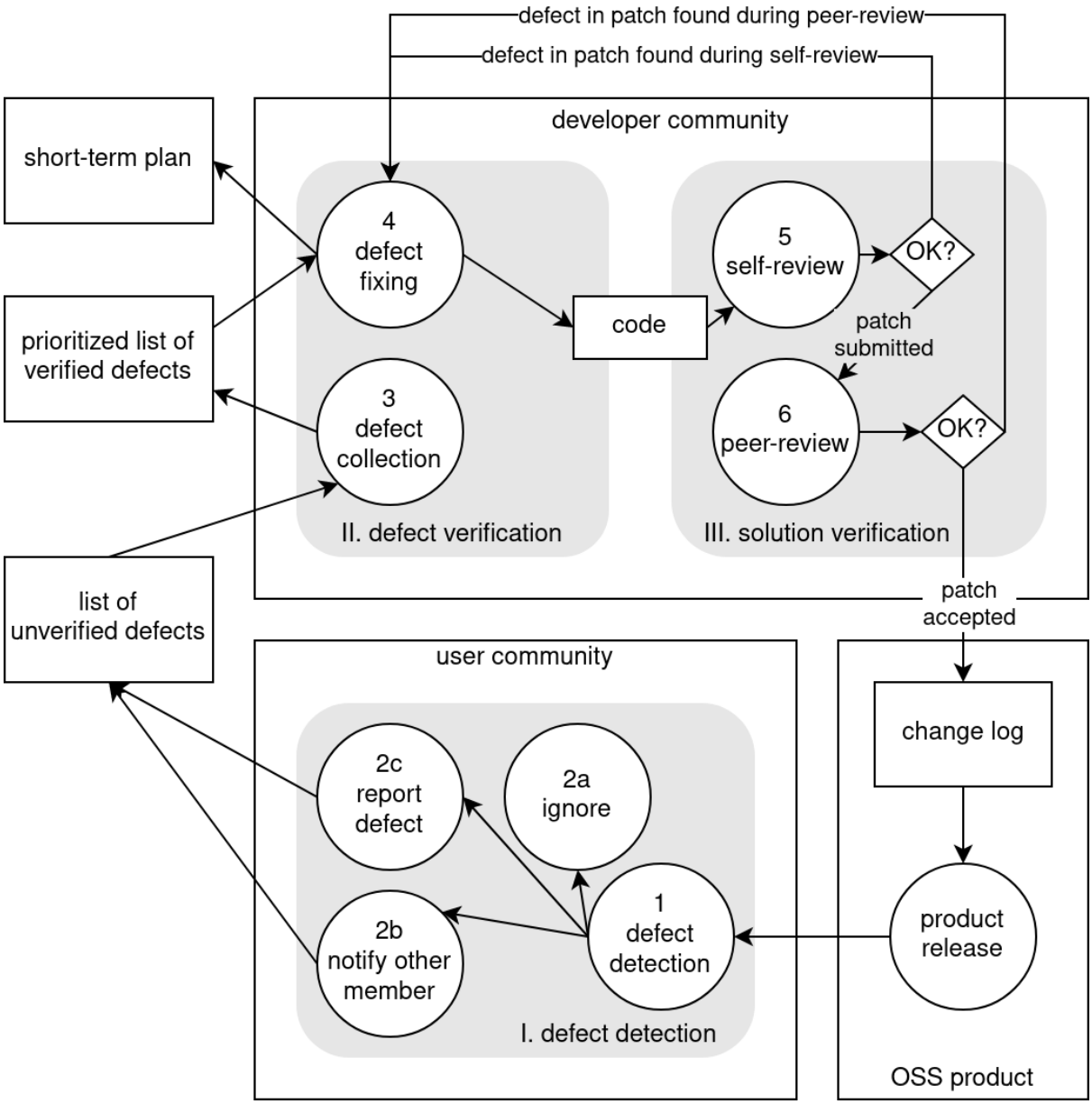


Figure 2.2: Quality Assurance activities under the Open Source Software Development model. Graphic based on [15, p. 4].

Chapter 3

Background

As established in the previous chapter, QA within the OSSD comes with a unique set of challenges and opportunities not seen within the closed source development model. Volunteer developers usually prefer to work on "exciting" new features or fixing long-standing bugs that affect themselves, rather than "boring" tasks like writing and maintaining automated tests. Consequently, the burden of automated testing is shifted towards the core team, taking away their attention from other responsibilities. User testing can remedy this because it provides a meaningful task for community members who do not have the necessary skills to contribute in other areas. Since there is no suitable free tool to manage system tests, however, "test runs are [often] managed using text processors or spreadsheets" [16, p. 1]. This practice significantly reduces the data's usefulness, as its analysis may require manual steps.

This chapter introduces the UBports Community as a reference OSS community. Three exemplary UBports software projects are described to illustrate the need for QA and establish use cases. Next, two existing user testing software solutions are explored regarding their fitness for these use cases. Building on this, a blueprint for a new software solution is outlined. In the following, this software will be referred to as OPEN-CUTS, the open crowdsourced user-testing suite.

3.1 Motivation: Ubuntu Touch and the UBports Foundation

The *Ubuntu*[1] operating system developed by *Canonical Ltd.* is one of the most popular *Linux distributions*[2] for servers and personal computers. A version for phones and tablets, later named *Ubuntu Touch*, has been in development since 2013 [17, p. 205]. Being "a phone operating system [that is] running the same core software as Ubuntu on the desktop" [17, p. 206], "Ubuntu Touch offers a totally extraordinary approach to deal with utilizing our mobile phones or tablets" [18, p. 578]. The user interface can adapt to different screen sizes and transforms into a desktop PC when connected to an external display, as seen in Figure 3.1.

[1] https://ubuntu.com

[2] Linux is a free- and open source Unix-like operating system kernel, see https://kernel.org. Operating systems that use the Linux Kernel are typically referred to as Linux distributions.

Figure 3.1: Ubuntu Touch on different screen sizes. Graphic (c) 2015 Canonical Ltd.

Ubuntu Touch achieved critical acclaim early on, with CNET declaring it "the most exciting thing at Mobile World Congress [a major technology trade show]" [19] in 2013, but failed to capture a significant market share to become a commercial success. Development leadership of the operating system was consequently transferred to the hands of the volunteer-led *UBports* Community in 2017 to allow Canonical Ltd. to focus on expanding its cloud-services and IoT departments [18, p. 579], [20], [21]. The community has been able to secure donations and established a non-profit foundation under German civil law [22] to provide long-term sustainable funding for the continued development of the operating system. A small core team to manage the community and handle essential development tasks is employed by the foundation [23]. However, as common with OSSD projects, a considerable part of the work continues to be done by unpaid volunteers from the community [23], [24]. As of January 2022, Ubuntu Touch runs on 78 different phones and tablets[3]. External hardware vendors provide commercial support for three of these devices, with the rest being maintained by volunteer contributors to the Halium[4] hardware enablement stack [24].

3.1.1 Community Structure

The structure of the UBports Community resembles the OSSD onion model discussed in section 2.3. However, due to the large number of different software projects associated with the Ubuntu Touch operating system[5], partially intersecting sub-communities

[3]https://devices.ubuntu-touch.io

[4]https://halium.org

[5]As of January 2022, the UBports GitHub organization lists 484 repositories, see https://github.com/ubports/.

emerge. Each subcommunity constitutes its own “onion within the onion” and maintains its distinct governance structure while contributing to the broader community’s goal.

3.1.2 Use Cases

This subsection discusses three example systems that illustrate the need for QA in the UBports Community.

3.1.2.1 Ubuntu Touch

Ubuntu Touch uses image-based upgrades with three regular channels: DEVEL for daily builds, RC (short for “release candidate”) for weekly builds, and STABLE for major versions [25]. Community members install these updates on their devices and report any issues they encounter in the public bug tracker [26]. A smoke testing plan for formalized user-testing of all operating system features (see Figure 3.2) that is “usually run on all devices before a new release to make sure no new issues were introduced” [26] exists. However, the growing number of supported devices creates a prohibitively expensive management overhead.

Storage - Ubuntu Touch Somoke testing

No.	Test	Expected Result	Actual Result
	Device is able to recognize microSD cards		
TC-0	1. Insert a new microSD	1. A notification about a new sd card appears on Device	PASS
	MicroSD card can be formatted		
TC-1	1. Insert a new microSD 2. Open External drives 3. Tap format button	1. A notification about a new sd card appears on Device 2. MicroSD card is listed 3. MicroSD card can be formatted	PASS
	MicroSD card can be safely removed		
TC-2	1. Insert a new microSD 2. Open External drives 3. Tap safely remove button	1. A notification about a new sd card appears on Device 2. MicroSD card is listed 3. MicroSD card can be safely removed	WONKY – Button needs to be tapped twice
	Storage can be accessed from a computer		
TC-3	1. Insert a new microSD 2. Connect device to a computer	1. A notification about a new sd card appears on Device 2. The computer can access both internal storage and ca	FAIL

Figure 3.2: Excerpt of the Ubuntu Touch smoke testing spreadsheet.

3.1.2.2 UBports Installer

The UBports Installer[6] is a graphical installation tool for Ubuntu Touch and other mobile operating systems. While being reasonably small in terms of lines of code, testing the UBports Installer proves very challenging. The primary complexity stems from the installer’s ability to run on the three different operating systems, Linux, macOS, and Windows, and each operating system can run different packaged versions of the installer. Additional complexity is added by accounting for the installable devices, as installation procedures vary significantly from one device to another. With a growing number of devices and operating systems supported by the installer, tracking all edge cases manually before every release can become very time-consuming.

[6] https://github.com/ubports/ubports-installer

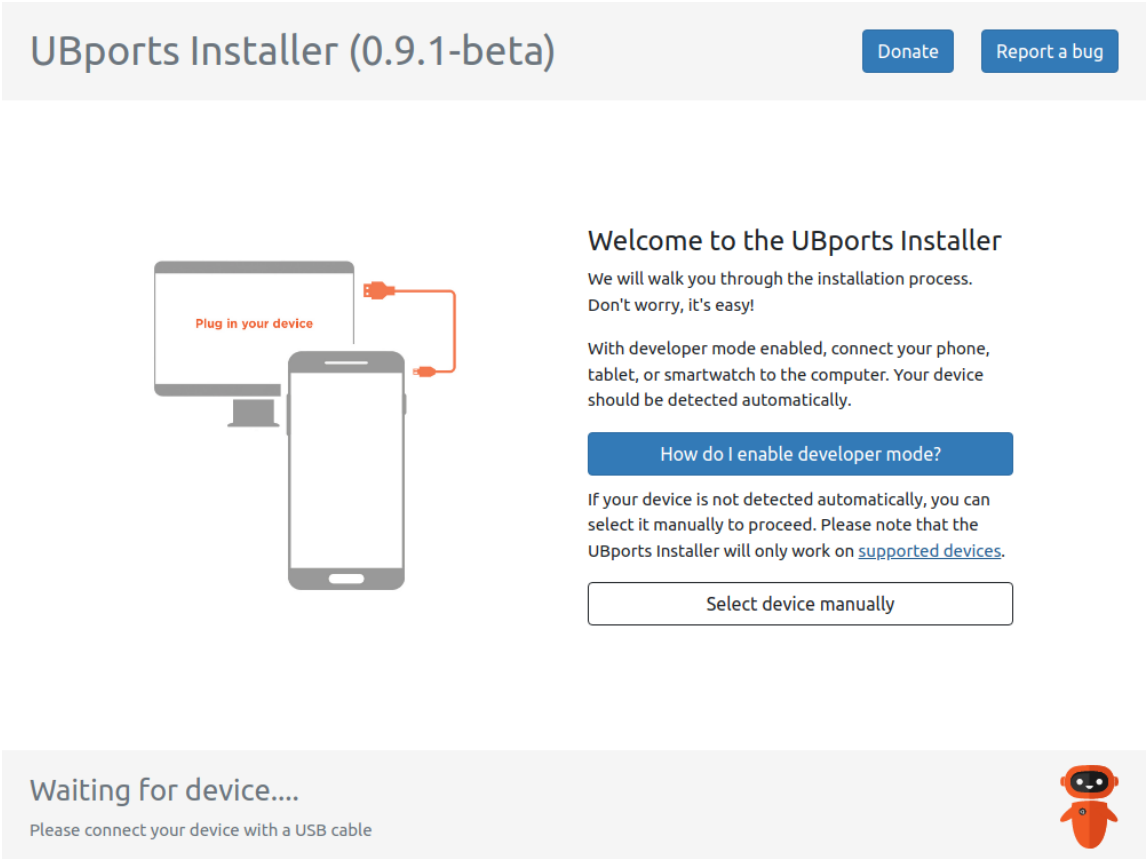


Figure 3.3: Screenshot of the UBports Installer.

3.1.2.3 8192

8192[7] is a small native puzzle game for Ubuntu Touch that can be downloaded and installed from the OpenStore[8]. It serves as an example of a small software project without specific requirements.

3.1.3 Discussion

The diversity of the community structure seen within the UBports Foundation makes it a promising candidate for an archetypical OSS community. The three example use cases were selected to represent the full range of the software complexity spectrum. Ubuntu Touch itself, being a vast and complicated system with countless features to test, poses a particular challenge for formalized user testing due to its large amount of test cases. The UBports Installer, while only requiring a single test case, comes with a wide array of edge cases that need to be covered. Testing solutions also have to account for simple projects like 8192 to avoid overcomplicating such systems. As such, a solution tailored to the UBports Foundation's needs should also fit most other OSSD projects.

[7] https://open-store.io/app/8192.neothethird

[8] The primary distribution channel for third-party applications designed for Ubuntu Touch, see https://open-store.io/.

3.2 Similar projects

3.2.1 Kiwi TCMS

Kiwi TCMS[9] aims to provide a fully fledged test-case management system that interfaces with bug trackers and external systems. The software is written in Python and provides a web interface and different APIs. A plugin system allows the creation of automated tests in various environments. Even though the project has been around since 2009, it has seen steady development and improvements. Kiwi TCMS does not provide combination logic, but the properties of a tested system can be recorded. Its user interface is tailored for professional users and requires some technical knowledge and familiarization. Figure 3.4 shows a screenshot of the user interface.

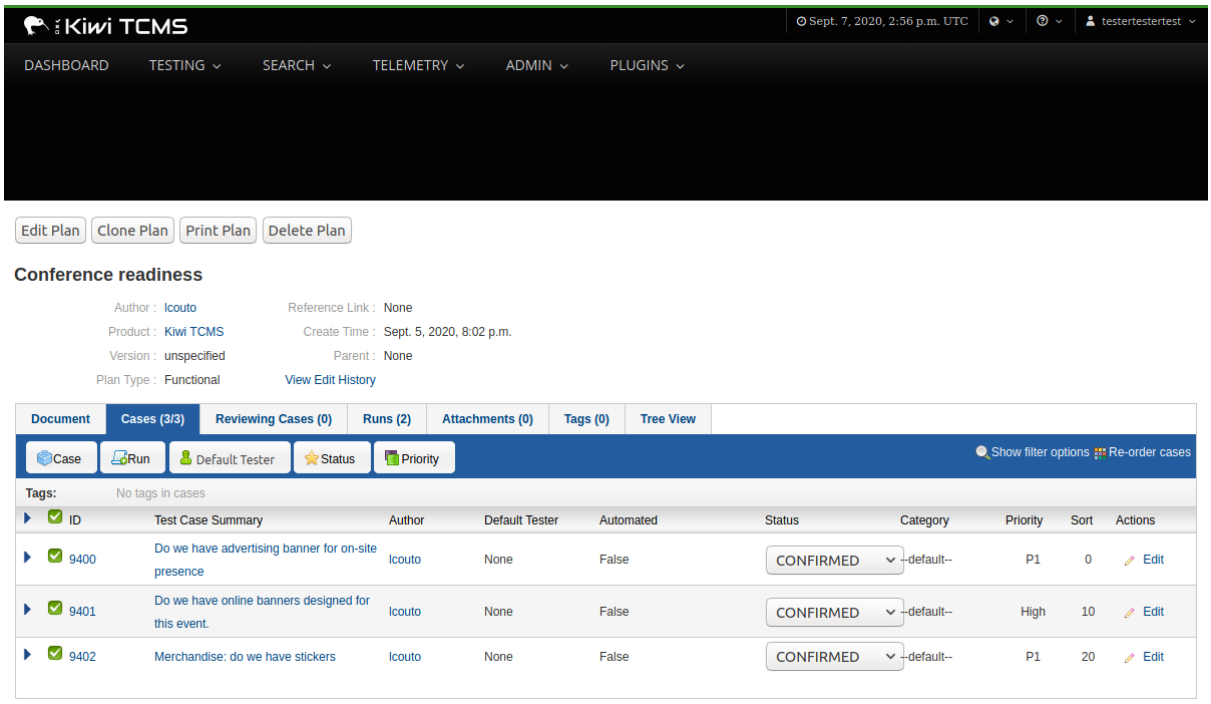


Figure 3.4: A test plan in Kiwi TCMS.

3.2.2 System Test Portal

System Test Portal[10] is a user-testing platform intended to be accessible for inexperienced users. The software is written in the Go programming language to make it lightweight to run. It allows organizing tests in test plans and has moderation features that allow users to comment on test runs to discuss the results. The main goal of the software is not to facilitate user testing as part of the day-to-day development process, but rather to provide a platform for end-users "to find out which features of a product were tested in which environment" [16, p. 1] to allow them to assess a software product, e.g., before purchasing supported hardware. The project was initially developed by students and faculty members of the University of Stuttgart in 2017. Unfortunately, as of January 2022, the repositories have not been updated in more than three years. System Test

[9] https://kiwitcms.org/
[10] https://www.systemtestportal.org/

Portal does not provide combination logic. Instead, a list of "variants" can be statically defined. Figure 3.5 shows a screenshot of the user interface.

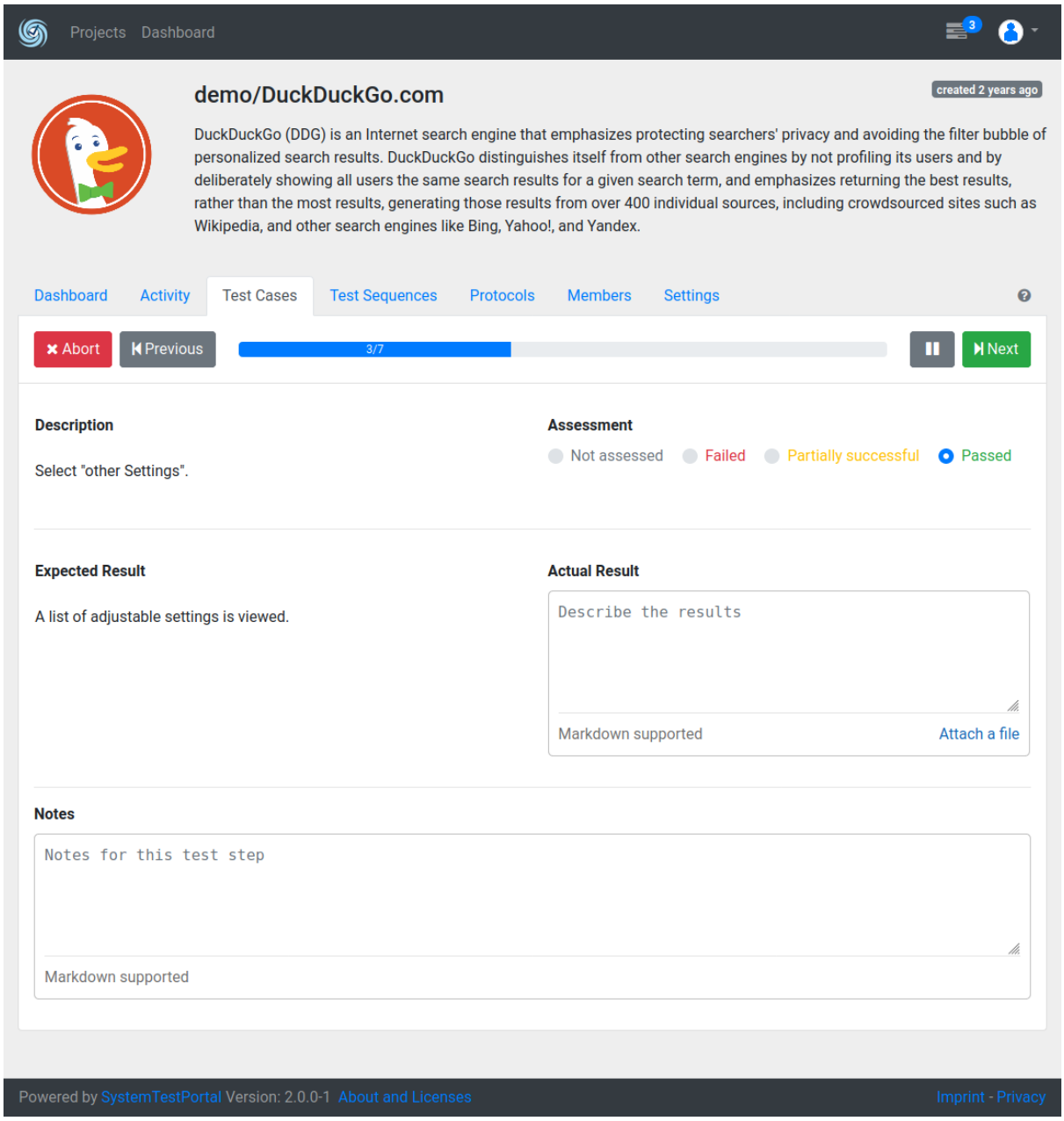


Figure 3.5: Running a test in System Test Portal.

3.2.3 Discussion

Even though both discussed projects have ostensible merits, they do not quite seem to fit the requirement set. While Kiwi TCMS is incredibly powerful, its professional interface is too overloaded to attract inexperienced community members. System Test Portal has an advantage here, but its contribution process is also cumbersome, as it is not intended for daily use. Neither solution is suitable for "drive-by" contributions. Additionally, both projects lack the combination filtering capabilities to represent the complex edge cases seen with the UBports Installer (see section 3.1.2.2).

3.3 Blueprint

OPEN-CUTS is an **open c**rowdsourced **u**ser-**t**esting **s**uite. It provides a human computation platform to facilitate formalized user testing for software developed under the OSSD.

3.3.1 Terminology and Structure

A piece of software tested with OPEN-CUTS is called a **system**. Releases or versions of the software are called **build**s. Every system defines **tests** and **properties**. A **user-generated** report of a test is called a **run**.

3.3.2 Feature Overview

An instance of OPEN-CUTS can track multiple systems. Volunteer community members can follow detailed instructions to run tests on different builds of these systems and record their results with the suite. Aggregated test results are automatically analyzed to provide meaningful results to contributing developers. OPEN-CUTS can integrate with continuous-integration and -delivery (CI/CD) pipelines to promote sufficiently tested builds automatically. Systems can be configured to automatically fetch new builds from external distribution channels to reduce administration overhead.

Systems can account for complexity by defining *properties* that testers need to specify when reporting their results. Valid property combinations can be specified using a simple yet powerful first-order logic language. The possible edge cases can be represented as a tensor consisting of all valid property combinations. Defect collection is aided by highlighting tensor slices of failing combination ranges. The tensor can be compared across subsequent releases to identify regressions.

Gamification features make the testing engaging. A karma system incentivizes recurring high-quality contributions and punishes vandalism. This system also allows users to unlock additional permissions on their own, thus minimizing the need for external moderator intervention by creating a self-sustaining meritocracy.

OPEN-CUTS exposes a public API[11] for all functionality. An optional default web interface that relies on this API is available, but implementing third-party clients would be feasible.

The API also allows for sourcing real-world testing data directly from systems in everyday operations. For example, the UBports Installer could prompt the user for consent to submit feedback at the push of a button. All required data (host operating system, package type, build number, logfile) can then be automatically read and submitted to the API by the software without requiring any additional user action. This approach will be primarily interesting for activities that users typically only perform once in a while. For example, while prompting feedback after using the installer is not a significant disturbance, users would not want to be interrupted multiple times a day from everyday activities.

[11] Application Programming Interface

3.3.3 User Interface

The primary objective of OPEN-CUTS' user interface design is to make reporting approachable for inexperienced users without oversimplifying the result presentation. The following sections describe the primary "views" in the user interface and their purpose.

Every view includes the same header and footer with essential navigation elements and links. The header will show navigation elements to create a new account or log in for a new user, as depicted in Figure 3.6. Logged-in users will see an option to manage their accounts instead, as depicted at the top of Figure 3.7. Administration UI elements, such as buttons to create, update or delete systems, tests, and builds, are only visible to users with the required permissions. Hovering over UI elements with the cursor reveals tooltips with usage information.

Figure 3.6: The header navigation for an unauthenticated user.

3.3.3.1 Dashboard

The dashboard is the main entry point and serves as the application's landing page, as depicted in Figure 3.7. The most prominent UI elements on the landing page are the OPEN-CUTS logo and a big blue button inviting the user to run tests in the center of the screen. Clicking either opens the reporting view described in section 3.3.3.2. The landing page also provides an overview of the systems available for testing and the registered users.

3.3.3.2 Reporting View

The reporting view provides a user-friendly interface for quickly reporting test runs. When starting from scratch, the user will first select a system and a build to test (Figures 3.8 and 3.9). If the system uses channels, the builds are grouped by channel. The view highlights those builds that have not yet reached sufficient test coverage and have not yet been tested by the logged-in user. As soon as a build is selected, the user can select a similarly highlighted test (Figure 3.10).

The user is then presented with detailed test instructions and an interface to report the test run metadata. The metadata consists of

- the system properties (such as the Ubuntu Touch device or the environment and package of the installer they are using),
- an optional comment, where the user can describe any observations they have made,
- any logfiles or debugging output generated by the system during test execution,
- and the result of the run.

OPEN-CUTS demo

Report My account Help

Welcome to OPEN-CUTS demo

open crowdsourced **u**ser-**t**esting **s**uite by Johannah Sprinz

Test something!

Systems

3 systems can be tested with OPEN-CUTS demo

System

System	Description
Ubuntu Touch	The worlds greatest mobile operating system, brought to you by the UBports Foundation!
UBports Installer	A graphical Installer for Ubuntu Touch and other alternative operating systems
8192	A 2048 clone for Ubuntu Touch

Users

3 users are testing with OPEN-CUTS demo

Name	Joined	Activity
Admin	7/13/2020, 3:29:50 PM	0
Johannah Sprinz	7/13/2020, 6:54:44 PM	0
Somebody	7/13/2020, 7:02:13 PM	0

OPEN-CUTS is free software licensed under the terms of the GNU AGPL version 3.0 or later.

OPEN-CUTS is still **very much in development**. If you're even *thinking* of using this in production, you're *crazy*.

(c) 2021 Johannah Sprinz privacy policy source code help

Figure 3.7: The OPEN-CUTS landing page for a logged-in administrator.

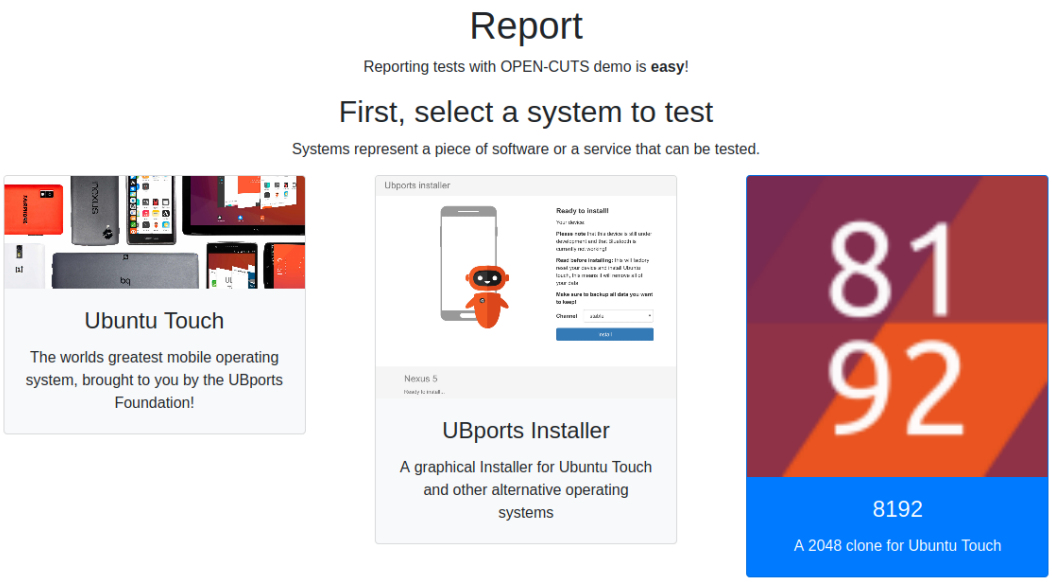


Figure 3.8: System selection in the reporting view. 8192 is currently selected, so it is highlighted in blue.

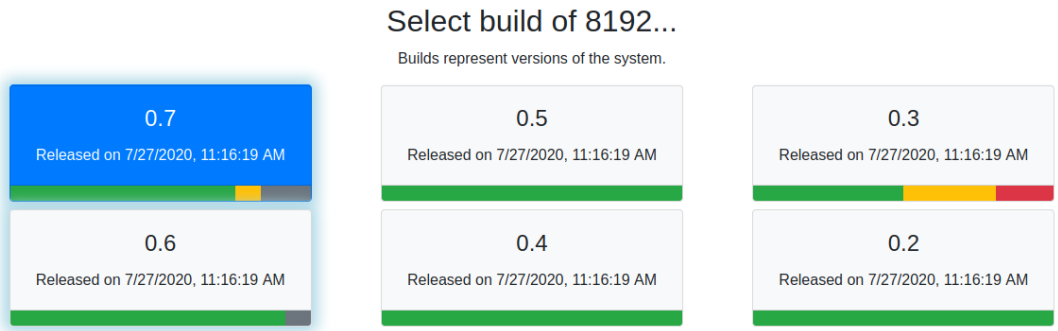


Figure 3.9: Build selection in the reporting view. Notice the different status indicator bars below every test. Builds 0.6 and 0.7 require the users' attention and are highlighted with a drop shadow.

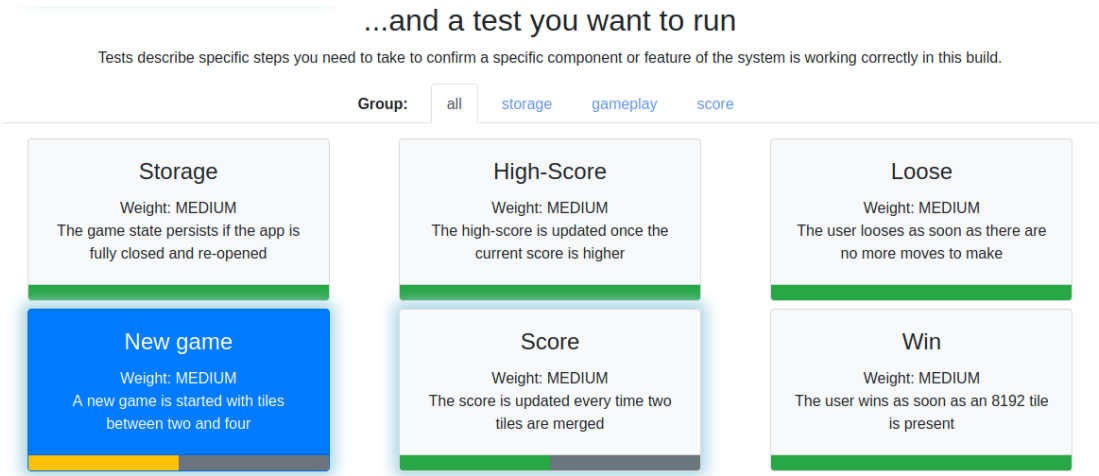


Figure 3.10: Test selection in the reporting view. Tests can be filtered by functionality group. The tests "New Game" and "Score" require the users' attention and are highlighted with a drop shadow.

Last but not least: Run the test "Install a device" on UBports Installer (0.4.18-beta)

Test Description Install an OS

Steps Connect a device to the installer. If the device is in developer mode, it should be recognized automatically. If not, you should be able to select a device manually.

Follow the on-screen instructions to finish the installation.

A couple minutes after the installer shows the end screen, your device should reboot into the new operating system.

Environment

What operating system are you using the installer on?

Package

What package of the Installer did you use?

Comment Everything works!

Anything else the maintainers should know

Logs Add log file | Add pasted log

Logs are automatically generated text files containing debugging information

Submit result PASS | WONKY | FAIL

Select a test result and submit

Please specify your system properties!

Figure 3.11: The submission form in the reporting view. The report can be submitted after entering the required metadata.

SYSTEM UNDER TEST

UBports Installer

Delete | Update | Build | Test | Property | Run

Builds

6 UBports Installer builds are available for testing

Tag	Date	Channel	Status
0.4.18-beta	4/3/2020, 3:59:47 PM	default	
0.4.17-beta	4/1/2020, 11:47:23 PM	default	
0.4.16-beta	3/23/2020, 7:52:22 PM	default	
0.4.14-beta	12/10/2019, 9:18:05 PM	default	
0.4.13-beta	12/6/2019, 4:04:53 PM	default	
0.4.12-beta	12/5/2019, 2:34:40 AM	default	

Tests

There are 1 tests defined for UBports Installer

Name	Description	Group	Weight
Install a device	Install an OS	default	CRITICAL

Ubports installer

Nexus 5

Testing UBports Installer : A graphical Installer for Ubuntu Touch and other alternative operating systems

Properties

CombinationFilter

Properties, eg. package types.

Name	Description
Environment	What operating system are you using the installer on?
Package	What package of the Installer did you use?

Figure 3.12: The system view for the UBports Installer seen by an administrator. A regular user can only see the button to create a new run, not the administration buttons.

3.3.3.3 System View

The system view (Figure 3.12) is easily accessible by selecting a system in the dashboard. It presents an overview of all system metadata and displays advanced controls for logged-in administrators. Sortable tables list tests and builds associated with this system. Views of the tests, builds, and runs associated with this system use the same righthand sidebar to provide a consistent experience and easy access to essential information.

3.3.3.4 Build View

The build view (Figure 3.14) contains the most relevant information for developers, as it presents the aggregated test results. Navigation elements above the title provide quick access to the previous and subsequent build on the same channel and the system overview page. The title also contains a quick summary of the test results. In the depicted case, `STILL_FAILING` indicates that an issue that caused a previous build to fail is still present in this build. Below the metadata, a chart shows the overall build status of the previous and subsequent build on the same channel. The view also lists tests and property combinations and displays their status. Clicking one of the entries allows deeper introspection and provides access to detailed tensor slices, as seen in Figure 3.13. Individual runs reported for this build are accessible from the bottom of the page.

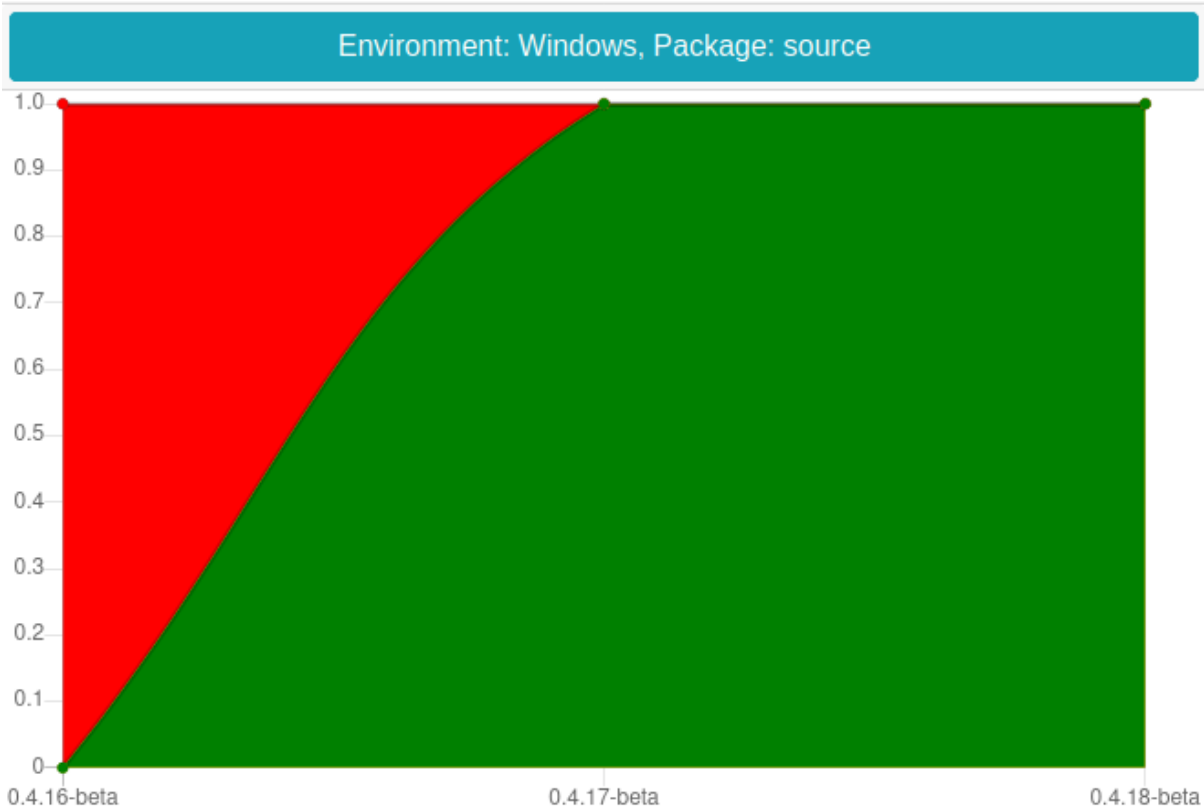


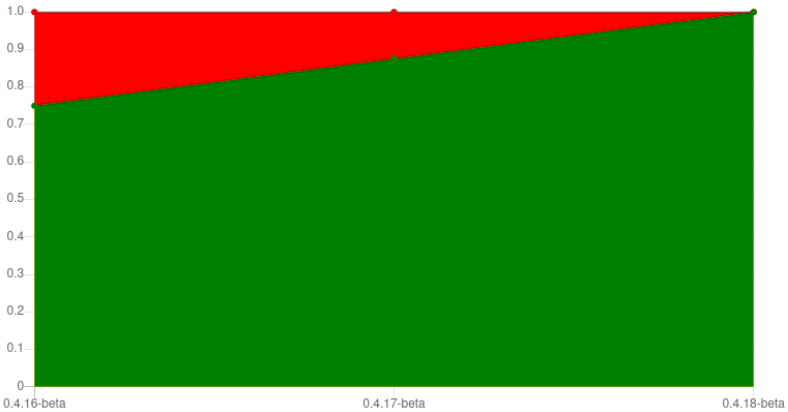
Figure 3.13: The tensor is sliced along a single test and property combination and visualized across three releases. When running a source package on Windows, the test was failing (red) in version 0.4.16-beta and passing (green) in the subsequent versions. Ergo the defect was successfully fixed.

← 0.4.16-BETA · UBPORTS INSTALLER · 0.4.18-BETA →

0.4.17-beta STILL_FAILING

Delete Run

- Released on 4/1/2020, 11:47:23 PM
- Channel: default



Tests

Name	Weight	Status
Install a device	CRITICAL	

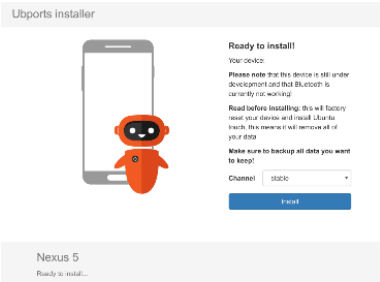
Combinations

Environment	Package	Status
Linux	snap	
macOS	source	
Linux	AppImage	
Windows	exe	
Linux	deb	
Linux	source	
macOS	dmg	
Windows	source	

Runs

There have been 16 runs reported for this build

Date	User	Test	Result
9/4/2020, 2:29:31 PM	*Anonymous*	Install a device	PASS



Testing UBports Installer : A graphical Installer for Ubuntu Touch and other alternative operating systems

Properties

CombinationFilter

Properties, eg. package types.

Name	Description
Environment	What operating system are you using the installer on?
Package	What package of the Installer did you use?

Figure 3.14: The build view for version 0.4.17-beta of the UBports Installer. The test "Install a Device" is failing (red) when running as a snap package on Linux and passing (green) for all other valid property combinations. The combinations list allows localizing this defect at a glance.

3.3.3.5 Run View

The run view (Figure 3.15) allows a more in-depth inspection of individual test runs. The view displays any logfiles and comments specified by the reporting user and metadata associated with the run. Navigation elements to return to the corresponding system, test, or build. Only moderators can see logfile contents from other users for security reasons.

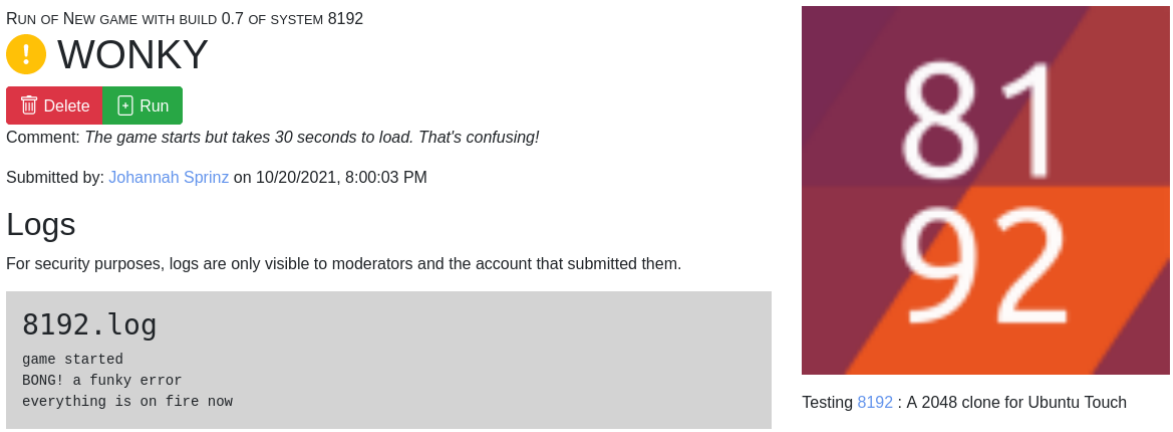


Figure 3.15: The run view for an individual test run on 8192. The tester added a comment describing their experience and included a logfile.

Chapter 4

Implementation

This chapter discusses technical implementation details of a prototype of OPEN-CUTS based on the blueprint outlined in section 3.3. Particular attention is paid to explaining the choice of third-party libraries, frameworks, and tools to meet the specified requirements.

OPEN-CUTS breaks down into three components: A client (the web interface being the default client), an API server, and a database. The web interface relies on this API exclusively and does not require direct database access. Figure 4.1 illustrates this communication structure.

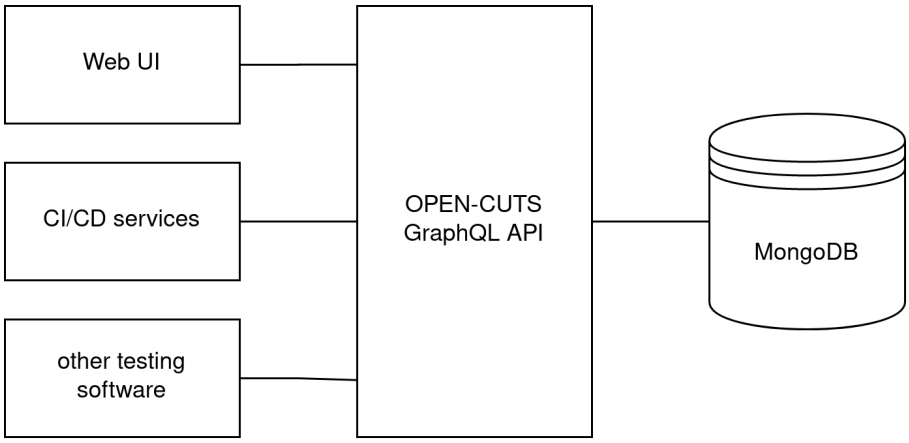


Figure 4.1: Communication structure between different clients, the API, and the database.

4.1 Database

OPEN-CUTS uses a *MongoDB*[1] database, as it is more efficient for data aggregation and associative data analysis than MySQL and other relational database implementations [27]. MongoDB can store rich *JSON*[2] data, which makes it highly flexible and very useful for the type of data OPEN-CUTS needs to store and analyze. An individual database

[1] An open source document-model database, see https://www.mongodb.com/.

[2] JavaScript Object Notation, "a text syntax that facilitates structured data interchange between all programming languages" [28, p. 5]

entry is called a *document*; documents of the same type are organized in *collections*. Every document is automatically assigned a unique identifier (`_id`).

The database schema breaks down into five collections representing the structure outlined in section 3.3.1:

- **users**: A user of OPEN-CUTS. The schema stores the user's personal information, login credentials, and moderation data.
- **systems**: A system is a piece of software or distribution of multiple functionally related pieces of software for testing with OPEN-CUTS.
- **builds**: Builds represent versions or releases of a system.
- **tests**: Tests contain instructions for users to validate a system's feature.
- **runs**: Runs contain test results reported by users.

See appendix 7.1 for the full commented database schema.

4.2 API server

The API server is written in *JavaScript* for the *NodeJS*[3] runtime. This chapter provides an overview of the design choices made and third-party technologies used.

The code is published under https://gitlab.com/open-cuts/open-cuts.

4.2.1 CRUDE GraphQL API

Thanks to MongoDB's powerful aggregation functionality, the database schema described in appendix 7.1 can efficiently be represented as a connected data graph, as depicted in Figure 4.2. This graph can be augmented with additional computed properties and traversed starting from different nodes. Graph representation has enormous potential for data processing and presentation, but designing an efficient *REST API*[4] for it can be challenging. This is because REST's capabilities of describing what data is required exactly are limited. Consequently, either the client will frequently have to make multiple requests to the API to resolve references, or the API will deliver data not needed by the client. This type of over- and under-fetching can cause time lag and network strain [31, p. 2228].

To avoid these problems, the OPEN-CUTS API uses GraphQL, "a query language designed to build client applications by providing an intuitive and flexible syntax […] for describing their data requirements and interactions" [32]. Instead of offering multiple endpoints that each return a fixed data structure, every request specifies its data requirements. GraphQL APIs typically only have a single HTTP endpoint that provides access to the entire graph. An API schema documents the structure of the available data. The following is an example query to fetch information about a single build in the data graph depicted in Figure 4.2. The returned information will include both data that can be read directly from that build's database entry, as well as data from connected nodes:

[3] JavaScript is a general-purpose programming language originally designed for web browsers. Nowadays, it is also widely used in servers via the NodeJS runtime, see https://nodejs.org/en/. JavaScript is specified and standardized as ECMAScript. The code for the OPEN-CUTS API server follows the ECMAScript 2019 standard [29] and is compatible with NodeJS version 14 or higher.

[4] Representation State Transfer. A popular standard for APIs on the internet [30, p. 62].

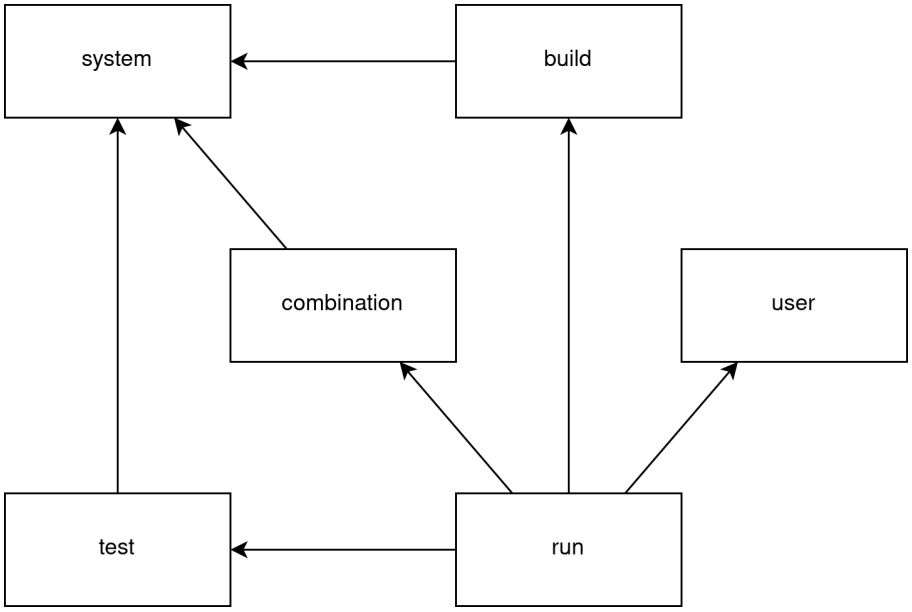


Figure 4.2: N-to-1 relationships allow representing the data as a connected data graph.

```
query {
  build(id: "42") {
    tag
    system { name description }
    runs {
      result
      comment
      user { name email }
    }
  }
}
```

The response will be a JSON object with a structure resembling the original query. See section 4.2.3 for information on how this query will be resolved. While a static data structure like this could still be depicted in a REST API, adding or removing fields would always require changes to the API server's code. GraphQL lifts that limitation; In theory, the entire graph could be traversed from any node using a single request. The API schema provides operations to create, read, update, delete, and explore data, thus implementing the *CRUDE* paradigm.

4.2.2 GraphQL Implementation

In addition to the official *graphql*[5] package, OPEN-CUTS uses a set of GraphQL libraries from the *apollo-server-express*[6] package to manage the API and integrate with the *express*[7] server framework. The *@graphql-tools/merge*[8] library is used to allow modularizing GraphQL schema definitions.

[5]https://www.npmjs.com/package/graphql
[6]https://www.npmjs.com/package/apollo-server-express
[7]https://expressjs.com/
[8]https://www.npmjs.com/package/@graphql-tools/merge

4.2.3 Data retrieval

Unlike REST-APIs, where data retrieval is usually hardcoded, GraphQL needs to resolve different data structures dynamically. To achieve that, GraphQL uses so-called resolver functions to resolve nested queries recursively. First, the root resolver is called, which is typically a database operation, a call to a microservice, or an external API. The data returned by the root resolver is matched against the fields requested in the query to omit unneeded data. The server then recursively calls field resolvers for every requested field to generate nested data. Before returning the data, it is matched against the API schema's data types to prevent unexpected results. [31, p. 2226f], [32]

Figure 4.3 visualizes how the server might resolve the example query given in section 4.2.1. The server begins by resolving the root query, which hits the build database with `_id: 42`. The subfields are then recursively queried by calling their resolvers in parallel. Resolver functions may also modify the data depending on the context; in this case, to prevent leaking email addresses to other users.

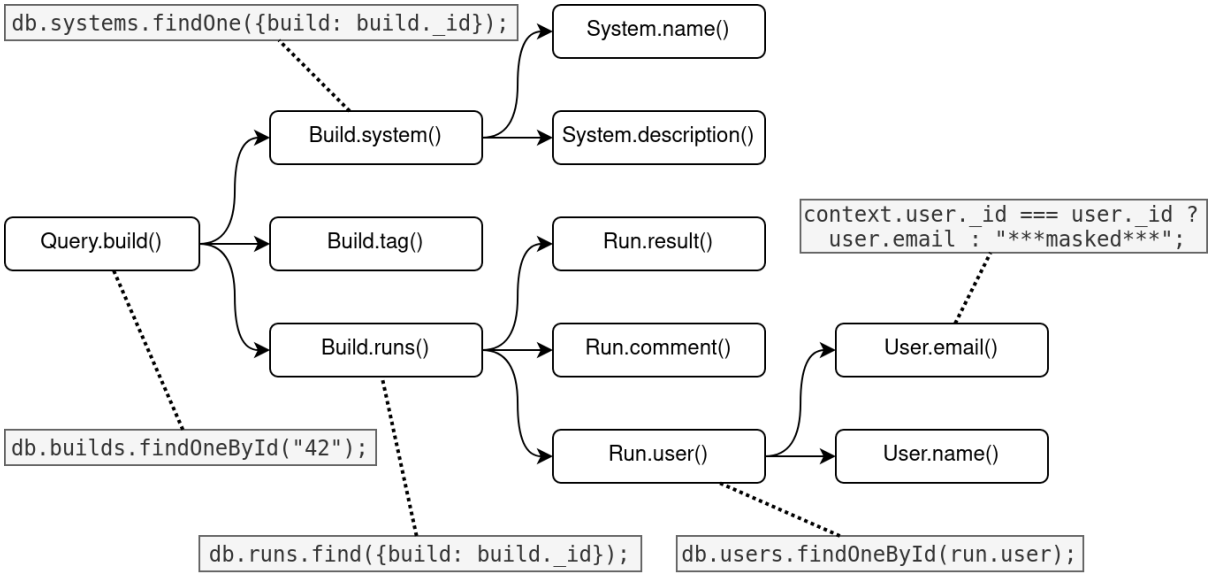


Figure 4.3: The GraphQL resolver chain for the example query given in section 4.2.1.

The above example demonstrates one issue GraphQL data resolvers need to account for: If one user reported multiple runs for the same build, the recursive resolver tree might hit the database multiple times to fetch the same user's information, which creates unnecessary strain. OPEN-CUTS prevents this by batching database requests sent in brief succession in the global scope and *deduplicating and caching*[9] database requests in the scope of a single query resolution. This was achieved by using the *mongoose*[10] MongoDB client behind a custom resolver system based on the *apollo-datasource*[11] and

[9]Caching is the practice of storing results of expensive operations so that future requests can be served directly from a store called *cache*. It is especially useful for small and unchanging data identifiable with a unique key, such as MongoDB's `db.<collection>.findById()` operation. Deduplication prevents duplicate requests with the same key, even if the older request is still pending. Both requests will finally be resolved simultaneously with the same data from a single operation. This practice is also known as memoization [33].

[10]https://www.npmjs.com/package/mongoose

[11]https://www.npmjs.com/package/apollo-datasource

dataloader[12] libraries.

4.2.4 Security

API queries can be authenticated and personalized by specifying a user's API token in the `authorization` HTTP header. If a valid token is specified, the server adds the corresponding user's database entry to the query's `context`, an object which is valid throughout query resolution. Figure 4.3 shows how the context object may be used to mask sensitive information such as other users' email addresses.

Since testers may report runs without logging in, OPEN-CUTS needs to filter out spam and other malicious content. The library *bayes-classifier*[13] implements a probabilistic classifier called the Naive Bayes Classifier [34, p. 210]. Every time a moderator flags a run as spam, the classifier learns. The classifier analyzes every new run and might hold suspicious entries for moderation.

The *nodemailer*[14] library is used to force users to confirm their email address after registration and allow password resets.

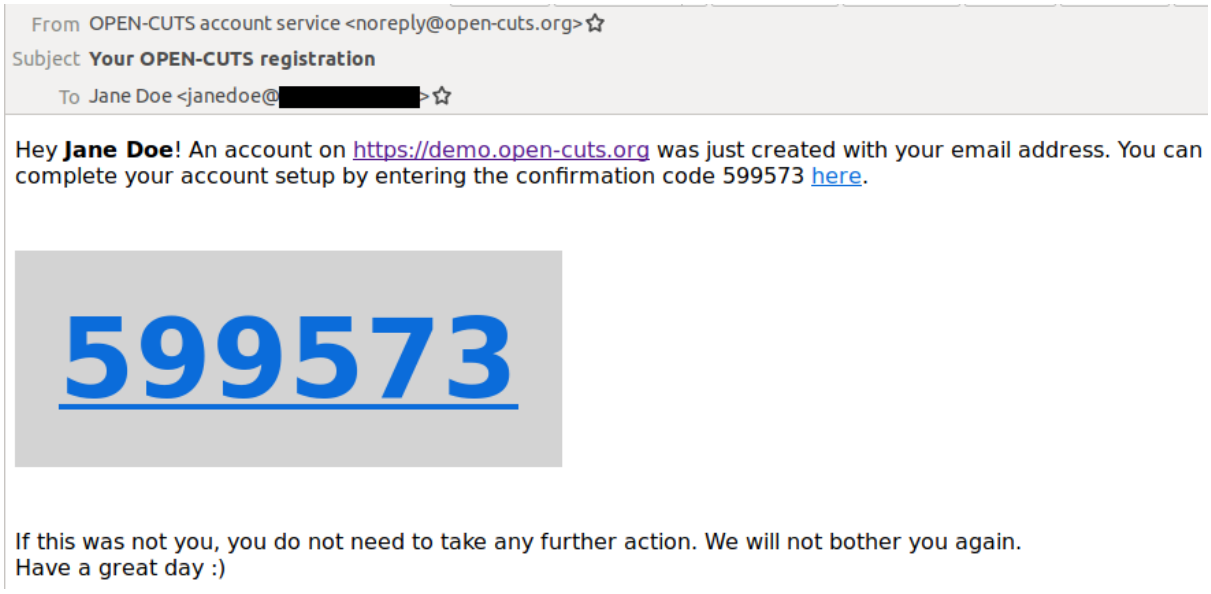


Figure 4.4: A confirmation e-mail sent by OPEN-CUTS.

The libraries *rate-limiter-flexible*[15] and *graphql-depth-limit*[16] protect the server from being overloaded with many concurrent or expensive requests.

4.2.5 Build discovery and retention

To allow long-term hands-off operation for moderators, OPEN-CUTS needs to discard old builds with their associated runs and discover new builds automatically. This discov-

[12] https://www.npmjs.com/package/dataloader
[13] https://www.npmjs.com/package/bayes-classifier
[14] https://www.npmjs.com/package/nodemailer
[15] https://www.npmjs.com/package/rate-limiter-flexible
[16] https://www.npmjs.com/package/graphql-depth-limit

ery and retention procedure is realized through a "housekeeping" pipeline, as depicted in Figure 4.5, which OPEN-CUTS runs once per hour.

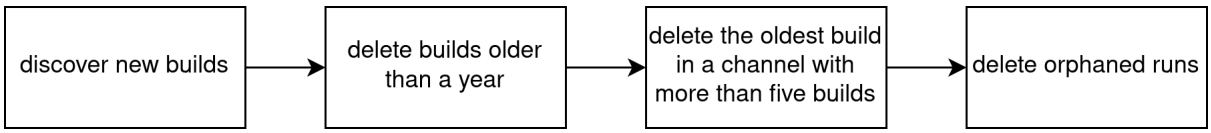


Figure 4.5: The housekeeping pipeline.

Based on the use cases described in section 3.1.2, there are three possible sources for build discovery: Ubuntu Touch uses its system image server, 8192 is published on the OpenStore, and the UBports Installer uses tagged GitHub releases. These services provide REST-APIs that OPEN-CUTS can query using the *axios*[17] HTTP request library. Every system can specify an array of string arguments used for build discovery (see appendix 7.1) to provide an agnostic and extendable configuration. The first element of the array specifies the build discovery engine, i.e., `systemimage`, `openstore`, `github`, or `gitlab`[18]. The remaining arguments specify the correct API endpoint. The API response is processed to return the five most recent builds per channel and filter out any build older than a year. If any of those builds do not yet exist in the database, they are created.

4.2.6 OPEN-CUTS Combination Filtering Language (OCFL)

The OPEN-CUTS Combination Filtering Language is a simple logical language to specify first-order logic expressions for filtering property combinations. The language was inspired by MongoDB's aggregation pipelines [35] and is a subset of JSON. Symbols include basic boolean algebra operators (`NOT`, negation, logical not, ¬; `AND`, conjunction, logical and, ∧; `OR`, disjunction, logical or, ∨; `IMPLIES`, implication, ⇒), as well as a `CHECK` function to test whether a property variable is in a set. The language is specified as follows:

```
exp       : <check> | <unaryexp> | <binaryexp> | <chainexp>
check     : { CHECK: [<stringterminal>, [<stringterminal>, ...]] }
unaryexp  : { <unaryop>: [<exp>] }
unaryop   : NOT
binaryexp : { <binaryop>: [<exp>, <exp>] }
binaryop  : IMPLIES
chainexp  : { <chainop>: [<exp>, ...] }
chainop   : AND | OR
```

Being a subset of JSON, most developers will already be familiar with the syntax and only need to learn the five operators. Additionally, JSON's validation and formatting tools can be used (see Figure 4.6). For example, any valid OCFL expression can quickly be formatted to a human-readable structure by running `JSON.stringify(JSON.parse(ocflExpression), null, "  ");` in any JavaScript environment without requiring any additional libraries.

Any system that specifies properties can also specify an OCFL expression to filter the combinations required to cover the system in all its complexity. The OCFL expression is

[17] https://www.npmjs.com/package/axios

[18] GitLab is a free- and open source competitor to GitHub, see https://gitlab.com. It provides an API almost identical to GitHub, so supporting it as well is relatively straightforward.

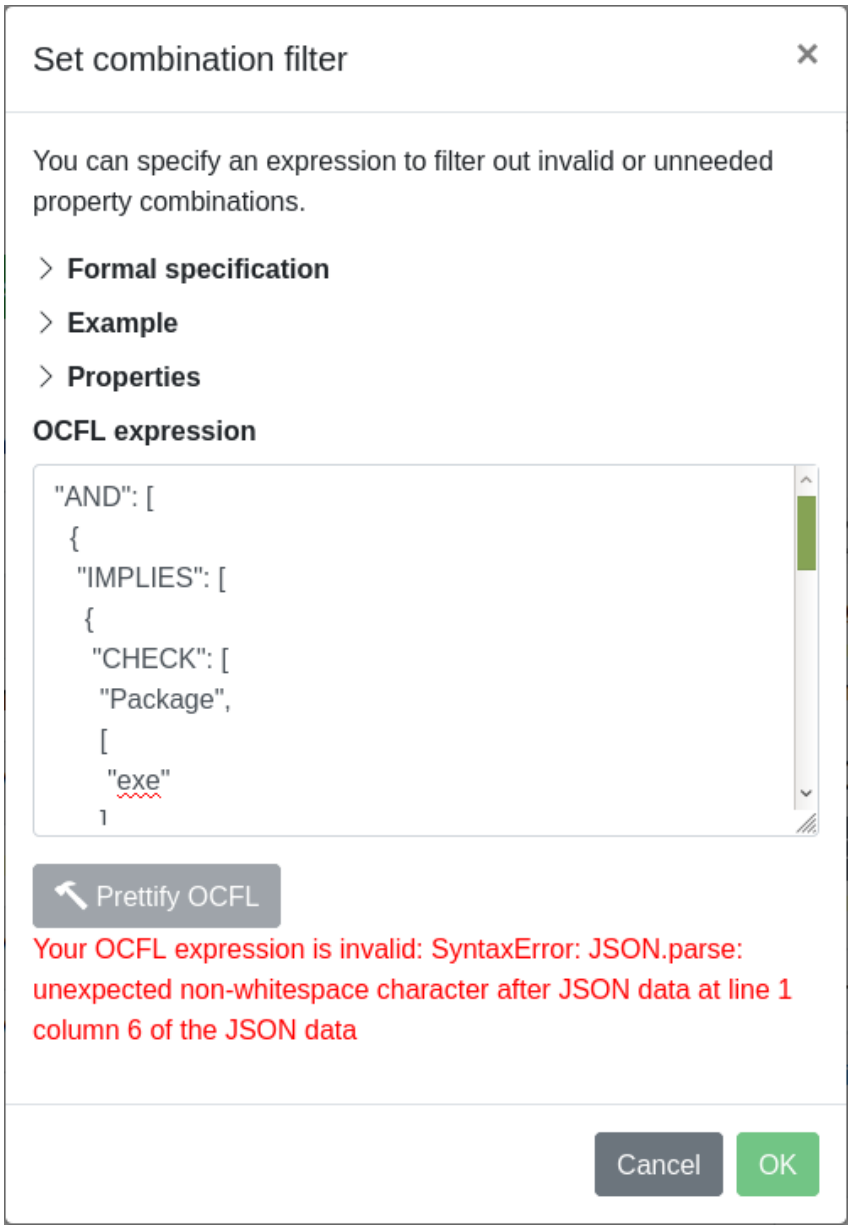


Figure 4.6: The OCFL modal in the web UI displaying a parsing error in red text. Only valid expressions can be saved, so the OK button is disabled while the error persists.

read from the database and compiled into a MongoDB aggregation pipeline expression. The following example generates all valid property combinations for the use case of the UBports Installer (see section 3.1.2.2):

```
{"AND": [
  {"IMPLIES": [
    { "CHECK": ["Package", ["exe"]] },
    { "CHECK": ["Environment", ["Windows"]] }] },
  {"IMPLIES": [
    { "CHECK": ["Package", ["dmg"]] },
    { "CHECK": ["Environment", ["macOS"]] }] },
  {"IMPLIES": [
    { "CHECK": ["Package", ["snap", "AppImage", "deb"]]},
    { "CHECK": ["Environment", ["Linux"]] }]
  }] }
```

This means that

- the `exe` package requires a `Windows` environment,
- the `dmg` package requires a `macOS` environment, and
- the `snap`, `AppImage`, and `deb` packages require a `Linux` environment.

Other combinations – such as the `source` package, which can run in every environment – are not affected.

4.2.7 Utility libraries

Other utility libraries include *debug*[19] for name-spaced logging, *dotenv*[20] for reading environment variables from `.env` files, *deep-equal*[21] for deep object comparison, and *uuid*[22] for generating secure tokens.

4.3 Web UI

The client is built using the *VueJS*[23] JavaScript framework with the *bootstrap-vue*[24] CSS library. It is implemented as a single-page application (SPA) with client-side routing to make page transitions more fluent. As such, instead of serving different HTML files for every page, the client consists of "a single HTML shell" [36] that is dynamically updated on navigation and user interaction. The libraries *vuex*[25] and *vue-router*[26] provide that functionality.

GraphQL API is accessed using the *apollo-client*[27] and integrated into the VueJS

[19] https://www.npmjs.com/package/debug
[20] https://www.npmjs.com/package/dotenv
[21] https://www.npmjs.com/package/deep-equal
[22] https://www.npmjs.com/package/uuid
[23] https://www.npmjs.com/package/vue
[24] https://www.npmjs.com/package/bootstrap-vue
[25] https://www.npmjs.com/package/vuex
[26] https://www.npmjs.com/package/vue-router
[27] https://www.npmjs.com/package/apollo-client

components using the libraries *vue-apollo*[28], *graphql*[29], and *graphql-tag*[30]. Caching and deduplication is achieved using *apollo-cache-inmemory*[31], *apollo-link*[32] and *apollo-link-http*[33]. Rich data presentation using the *chart.js*[34] implementation *vue-chartjs*[35]. The *core-js*[36] polyfill allows the client to use modern JavaScript features in older browsers.

The code is published under https://gitlab.com/open-cuts/open-cuts-web.

4.4 Administration

OPEN-CUTS can be quickly deployed to a production environment by running an *ansible playbook*[37]. Ansible is a server configuration utility that can read declarative config files ("Playbooks") for idempotent – that is, repeated execution with the same input will always yield the same result – operations. The playbook connects to an Ubuntu Server using *SSH*[38] and performs the following steps:

1. The OPEN-CUTS *git*[39]-Repositories are cloned to a directory on the server.
2. An *Nginx*[40] server configuration is prepared for use with the specified domain. A *Letsencrypt TLS certificate*[41] is obtained.
3. The three components that make up OPEN-CUTS - web server, API server, and database - are set up as containerized microservices using *docker-compose*[42], resulting in a structure as depicted in Figure 4.7. The previously configured Nginx server serves pre-built and optimized static HTML files for the web client and provides SSL termination for reverse-proxying API calls to the API server's container. The API server and database docker containers are only accessible from the Nginx and API server's containers.
4. If necessary, the database is initialized with sample data.

The playbook is published under https://gitlab.com/open-cuts/open-cuts-ansible.

[28] https://www.npmjs.com/package/vue-apollo
[29] https://www.npmjs.com/package/graphql
[30] https://www.npmjs.com/package/graphql-tag
[31] https://www.npmjs.com/package/apollo-cache-inmemory
[32] https://www.npmjs.com/package/apollo-link
[33] https://www.npmjs.com/package/apollo-link-http
[34] https://www.npmjs.com/package/chart.js
[35] https://www.npmjs.com/package/vue-chartjs
[36] https://www.npmjs.com/package/core-js
[37] https://docs.ansible.com/ansible/latest/user_guide/playbooks.html
[38] Secure Shell (SSH) is a protocol for secure networking that is commonly used for remotely controlling web servers [37].
[39] A free and open source distributed version control system widely used for managing source code in software projects, see https://git-scm.com/.
[40] A free and open source web server, see https://nginx.org/.
[41] Transport Layer Security (TLS, also often referred to by the name of its predecessor Secure Sockets Layer, SSL) is a protocol for secure data transmission over the internet [38]. TLS relies on encrypted certificates issued and cross-signed by certificate authorities. Letsencrypt is a not-for-profit certificate authority that provides free TLS certificates, see https://letsencrypt.org/.
[42] Docker is a system for lightweight operating-system-level virtualization "containers" that can be orchestrated and composed, see https://docs.docker.com/compose/.

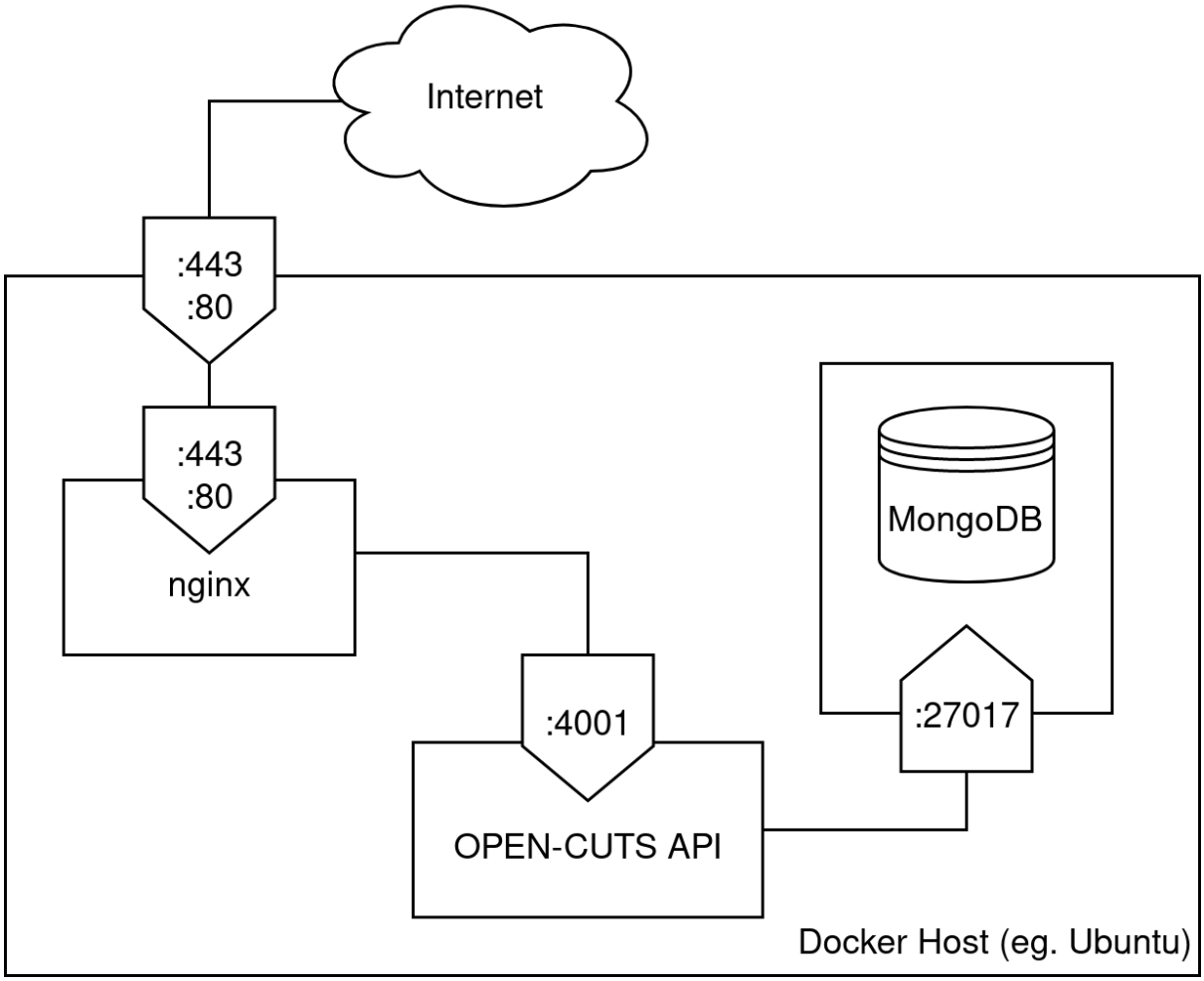


Figure 4.7: Microservices running as docker containers.

Chapter 5

Evaluation

This chapter outlines a method for evaluating the previously developed prototype of OPEN-CUTS. The goal of the evaluation is to answer three questions through a usability study:

1. Is there any initial evidence of untapped potential for facilitating QA under the OSSD by having inexperienced community members contribute formalized user testing data to a human computation platform?
2. Can OPEN-CUTS be efficiently used by the inexperienced members of an open source community to conduct and report formalized user tests?
3. Is OPEN-CUTS equipped to provide meaningful insights to open source developers in their daily work?

The study results are presented, analyzed, and interpreted to gain insights into possible future development goals and prepare for further (long-term) studies.

5.1 Methodology

Several scientific methods to assess the usability of software exist. So-called *think-aloud* methods require the subject to recount their experience, either while using the system (*concurrent think-aloud*) or while watching a recording of their journey through the system (*retrospective think-aloud*). *Probing* methods require the instructor to ask the subject targeted questions about their experience. While concurrent think-aloud is believed to be "the best methodology [...] for collecting real-time cognitive data during usability tests" [39, p. 2], combining it with *retrospective probing* seems to be "the most comprehensive and effective way to attain insight into users' experiences and understand a system's usability issues" [39, p. 4].

The retrospective probing section of this study consists of a questionnaire that includes statements that the subjects are asked to rate on a five-point Likert scale to measure attitude by offering "5 categories of response, from (for example) 1 = strongly disagree to 5 = strongly agree" [40, p. 1217]. The complete questionnaire is provided in section 5.2.4.

5.2 Trial

The study's goal is to assess the usability of OPEN-CUTS version 0.0.1[1], both for reporting and analyzing test results. For this, subjects were given a set of tasks. The first group (*user group*) was recruited from ordinary members of the UBports Community. The five subjects in this group are active users of Ubuntu Touch who are not involved in the development of core UBports projects. Subjects from the user group were given tasks that represent the workflow of a volunteer that wants to help test software with OPEN-CUTS. The second group (*developer group*) consists of four members of the core Ubuntu Touch development team. In addition to the tasks given to the user group, the developer group was presented with tasks that represent a developer's workflow. This includes triaging test results to locate and identify regressions, moderating test runs, and configuring systems in OPEN-CUTS.

5.2.1 Environment

The trial was conducted as a video call between the instructor and the subject. In order to mimic the circumstances of a typical volunteer contributor, the subject was asked to choose an environment where they can comfortably work without distractions.

The subject is seated at an internet-connected personal computer with the UBports Installer and a web browser installed. Subject and instructor can talk to each other and the instructor can see the subject's screen. The subject also has access to a fully charged Ubuntu Touch mobile device with the app 8192 installed. All tools installed on the subject's computer or mobile device may be used during the trial and the world-wide-web may be accessed to retrieve any information that might be useful.

Before beginning their tasks, the subject is briefed on the motivation behind OPEN-CUTS and the concepts described in section 5.1. They are encouraged to speak while completing their tasks.

5.2.2 Sample data

Each subject is confronted with an identical installation of OPEN-CUTS with the same data in the database. The database is meant to resemble an OPEN-CUTS configuration as it might be used in the UBports Community. The three systems described in section 3.1.2 - Ubuntu Touch, UBports Installer, 8192 - are available for testing with their latest releases fetched from their respective build discovery sources (see section 4.2.5). Every system is modeled with the required properties, combination filters, and tests. A set of runs is already reported to create the environment needed for the tasks described in section 5.2.3.

A BSON-snapshot[2] of the database is published under https://gitlab.com/open-cuts/sample-data.

[1] Published under https://gitlab.com/open-cuts/open-cuts/-/tree/0.0.1/ and https://gitlab.com/open-cuts/open-cuts-web/-/tree/0.0.1/

[2] Binary JSON is a standard for binary-encoded JSON-like data, see http://bsonspec.org/.

5.2.3 Tasks

The subjects are given a series of tasks. Think-aloud comments and completion times are recorded. Both groups must solve tasks 1 through 4, representing challenges users will typically face when testing software with OPEN-CUTS. They are meant to resemble the journey of a new unregistered user exploring the system, starting with a simple anonymous report (task 1) and then going through user registration (task 2). The third task requires the user to perform a more complex test and provide a logfile from their computer. To complete the fourth task, the user will have to read and interpret the test coverage results and run multiple small tests. The developer group is given four additional tasks that cover reading and understanding the accumulated test results (task 5), as well as the OPEN-CUTS moderation (task 6) and administration (tasks 7 and 8) workflow.

1. Find a test for Ubuntu Touch, perform the steps on your device, and report the result to OPEN-CUTS. You do not need to provide a logfile.
2. Create an account in OPEN-CUTS and log in.
3. Re-install Ubuntu Touch on your device using the UBports Installer and report the result in OPEN-CUTS. Provide a logfile of your installation.
4. Run all inconclusive tests on the latest release of 8192. You do not need to provide logfiles.
5. Examine the latest couple builds of the UBports Installer. Can you identify any regressions? If so, what property combination seems to be causing the problems? When were the regressions introduced, when were they resolved?
6. Examine the test runs for version 0.7 of 8192. A user seems to have accidentally submitted an invalid run with misleading comments. Identify and remove that run.
7. Add the Ubuntu Touch OpenMensa app[3] as a new system. Configure build discovery and at least one test for it.
8. Consider the following scenario: In addition to the Graphical user interface (GUI), a new optional command-line interface (CLI) that is only available on Linux is added to the UBports Installer and should be tested using OPEN-CUTS. Add a new property to record the usage mode and adjust the combination filter to calculate correct test coverage results.

5.2.4 Questionnaire

After all tasks have been completed, the subject is then confronted with statements and asked to rate their attitude on a five-point Likert scale (strongly disagree = 1, disagree = 2, neutral = 3, agree = 4, strongly agree = 5). Statements 1 through 6 reflect on tasks 1 - 4 and are answered by both groups; statements 7 through 11 are only shown to the developer group. To prevent subjects from answering in a pattern, statements 5, 8, and 10 have been negated.

1. I instantly understood the User Interface (UI).
2. I quickly found the UI elements I was looking for.

[3] https://open-store.io/app/openmensa.neothethird

3. The software provided helpful hints and explanations when I did not understand something.
4. I could imagine using the software regularly (i.e., at least once per week) to report tests.
5. The software is not suitable for beginners.
6. This software allows volunteers who do not know how to code to contribute in a meaningful way.
7. The insights provided by the software could help me in my daily work.
8. The software is oversimplified and cumbersome to use for experienced users.
9. The software can represent the system I develop in all its complexity.
10. I prefer direct user feedback over using formalized user testing.
11. The software makes it easy to identify regressions that only affect edge cases.

5.2.5 Criteria of Success

Every participant should be able to solve the tasks without help from the instructor. Non-breaking bugs[4] should be recorded as a think-aloud comment. Breaking bugs are not expected.

The time it takes to complete each task might be affected by external factors such as internet speed and other technical issues, so some variance is to be expected. Task three involves running a more complex test and might take up to fifteen minutes depending on the participant's internet connection speed. Ten minutes are allotted for task eight because it involves understanding a scenario and getting acquainted with the OCFL. Solving the other tasks should require significantly less time. The developer group should be able to solve most tasks slightly faster than the user group, as they might be more familiar with the terminology used.

Low values (disagreement) are desirable for the questionnaire statements 5, 8, and 10; high values (agreement) for the others.

5.3 Results and Discussion

This section presents the results of the study. It includes the time needed to complete the various tasks, the results of the questionnaire, as well as think-aloud comments from the participants.

5.3.1 Task Completion Times

The completion times for every task are presented using box plots in Figure 5.1. Raw data is available in appendix 7.3.

All participants were able to solve the tasks without help from the instructor. While the developer group solved all tasks on average a little faster than the user group, the

[4] A flaw in a software program is referred to as a bug. If the user cannot find a work-around to accomplish their task another way, it is referred to as a breaking bug.

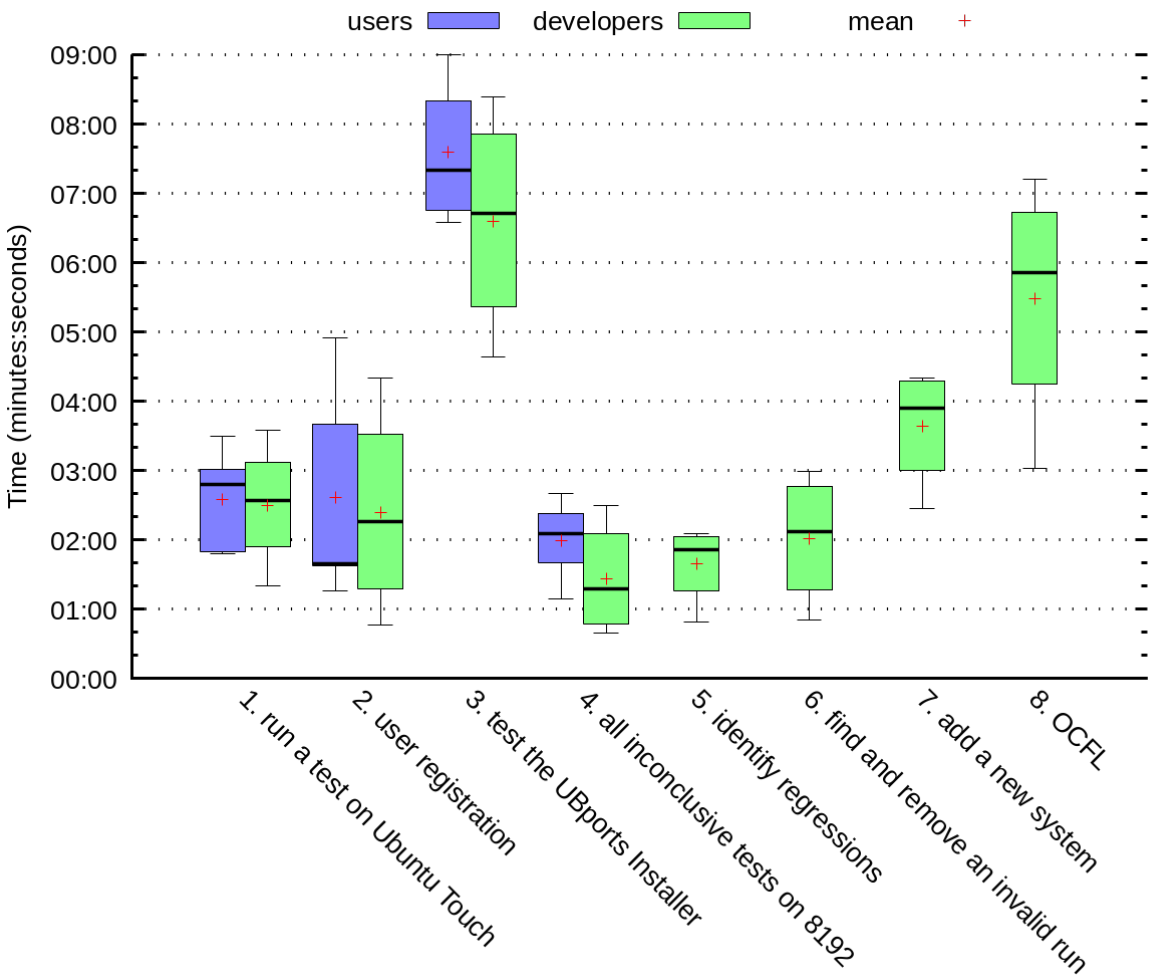


Figure 5.1: Time it took participants to complete each task. See section 5.2.3 for original task descriptions.

difference is never greater than one minute, with task three - using the UBports installer and providing a logfile - showing the most significant difference. This task was also the most time-consuming, with participants taking between 4:38 and 9:00 minutes to solve it. The second-most time-consuming task, which took participants between 3:02 and 7:12 minutes, was task eight - modifying an OCFL statement. Significant variance in duration was also observed in user registration (task two), which took between 0:46 and 4:55 minutes. All other tasks were solved in 4:20 minutes or less.

5.3.2 Questionnaire Results

The questionnaire results for every task are presented using box plots in Figure 5.2. Raw data is available in appendix 7.3.

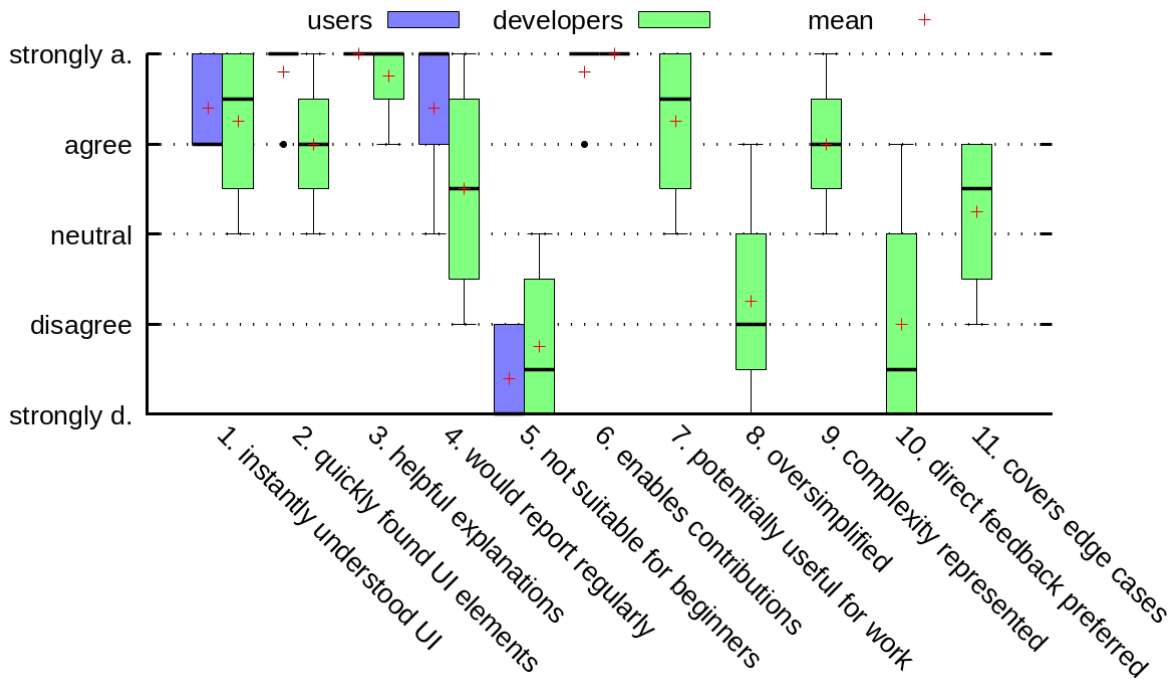


Figure 5.2: Questionnaire results. See section 5.2.4 for original statement descriptions.

Participants generally agreed to the first two statements on the questionnaire, which address the user-friendliness of the UI. The responses to these statements were slightly more scattered in the developer group, with one participant expressing a neutral attitude. All participants agreed or strongly agreed that the software provides helpful hints (statement 3). When asked whether they could imagine using the software regularly to report tests (statement 4), most participants agreed, with only one member of the developer group expressing disagreement. Users and developers alike viewed the software as suitable for beginners (statement 5) and agreed that the software helps in enabling contributions (statement 6). The developer group generally agreed that the software could be helpful in their daily work (statement 7) and that the systems they work on can be represented in all their complexity (statement 9). Most developers viewed the software as not oversimplified (statement 8) and stated that it makes identifying regressions that only affect edge cases easy (statement 11). While some developers preferred direct user feedback over formalized user testing (statement 10), most did not agree with that sentiment.

5.3.3 Think-Aloud Comments

The think-aloud method produced several valuable comments. The original comments are provided verbatim in appendix 7.4.

The comments were mostly positive, with participants commending the intuitive UI structure and legibility. The wish for an optional dark UI mode and the support for markdown[5] formatting in run comments and test instructions was expressed. One participant remarked that it is currently impossible to change administrator settings like the number of required runs per combination from the web interface. While completing task two - registration - it was lamented that the user needs to copy the confirmation code manually. Some websites offer a clickable confirmation link, which might save time. Another participant suggested implementing a single-sign-on service such as Ubuntu SSO[6].

A large amount of constructive feedback was given on the reporting workflow. A tutorial explaining the core concepts to new users was proposed. It was noted that manually providing logfiles is very time-consuming. One participant proposed recording the property combination before selecting a build and test to calculate the build coverage for the selected property combination individually. Similarly, highlighting of the missing property combinations was requested. Users expressed discontent that some test instructions were overspecified and that the reporting workflow was not engaging enough. Another point of criticism was that the reporting view always stores the last selected property combination. Instead - as the user suggested - the software should store multiple property combinations that can be selected quickly. It was also noted that the reporting view is missing a confirmation button; some users might prefer to select the result first, then write a comment and submit the run. One user remarked that the API is sometimes very slow when loading the reporting view.

The moderation functionality was criticized for being rudimentary, and several additional features were suggested, such as a moderation queue, comments on runs, and the option to edit submitted runs. Participants also suggested linking failing tests to external issue trackers and implementing logic to prioritize validating failing tests linked to issues across subsequent builds. The need for additional logic and visualization to make failing combinations and regressions easier was voiced. Logfile handling was lauded for providing security against leaking sensitive information.

The OCFL was well-received, with one participant calling it “nifty”. Among the minor additional features requested was the ability to comment OCFL statements, as well as syntax highlighting and matching brackets visualization in the editor.

The build discovery interface was criticized for providing insufficient documentation and visual cues about potential errors, which left the participants in the dark about whether they configured their systems correctly. It was suggested to implement a one-click configuration feature that makes educated guesses on most settings from a link to one of the build discovery sources to make adding new systems easier. One participant also suggested allowing individual builds to specify properties and property combinations that differ from the corresponding system’s properties. The rationale is that Ubuntu Touch might have different devices supported on different release channels of the system image server.

[5] A simple markup language with a focus on readability, see https://daringfireball.net/projects/markdown/syntax.

[6] https://login.ubuntu.com/+faq

No breaking bugs were discovered during the trial. The following non-breaking bugs were discovered:

- Runs with invalid combination options can be submitted, even though they do not contribute to the coverage.
- When using build discovery on the openstore, release dates are incorrect.
- Combination details can not be viewed on systems that only have a single test configured.

5.3.4 Discussion

No breaking bugs were discovered, and all participants were able to solve the tasks without help from the instructor, which suggests that the software is generally fit for use. The three non-breaking bugs discovered throughout the trial are related to data handling or UI code and should be easily fixed in a subsequent release of OPEN-CUTS. The tasks were generally solved quickly, with all participants staying well below the time limits defined for tasks three and eight. As expected, the members of the developer group were able to solve most tasks slightly faster than their peers from the user group, but the difference was never greater than one minute. Thus, the software can be efficiently used by amateurs and professionals alike. This was also reflected in the questionnaire answers: Participants expressed a positive impression of the user-friendliness of the UI (statements 1, 2). A majority of the participants from the developer group stated that the software is not oversimplified as to be cumbersome to use by experienced users (statement 8), so the priorities in UI design seem to have resulted in a good equilibrium.

Participants unanimously stated that the software is particularly beneficial for enabling contributions from users who do not know how to code (statement 6). Interestingly, the user group members were even slightly more likely to state that they would be willing to use the software regularly for testing (statement 4) and that the software was suitable for beginners (statement 5). This effect might also partly be because only the developer group was confronted with the software's more complex features. However, the results seem to support the original hypothesis of untapped potential in having inexperienced members of open source communities contribute to the QA process through formalized user testing in a human computation setting.

Participants from the developer group generally agreed that the software provides meaningful insights for their daily work (statement 7) and that it can sufficiently represent the complexity of the systems they work on (statement 9). However, a minority of participants from the developer group expressed a general preference for direct user feedback over formalized user testing. It is impossible to say whether this is because those developers have never experienced a practical formalized user testing workflow or because of an actual preference, but it would certainly be possible to keep channels for direct user feedback open next to a system like OPEN-CUTS, so this should not be an issue.

The think-aloud comments provided helpful feedback to improve the software, which has been compiled into a list of future development goals for OPEN-CUTS (see appendix 7.2).

Overall, the trial showed some initial evidence for the untapped potential of facilitating QA under the OSSD by allowing inexperienced community members to contribute to an

HC platform. The previously implemented prototype of OPEN-CUTS has proven fit for use and was very well received by the user study participants. Inexperienced users can efficiently run formalized user tests and report their results, and developers can draw useful information from the aggregated data for their daily work. A list of achievable future development goals has been created based on the insights gained in the user study. A future long-term study in a real-world open source community could be conducted to examine the benefits of using OPEN-CUTS. Particular attention should be paid to how this approach performs compared to standard QA practices.

Chapter 6

Conclusion and Outlook

This final chapter summarizes the contribution of this paper, outlines plans for future work, and concludes with a summary.

6.1 Contribution

This project set out to identify and address challenges faced by open source communities when doing quality assurance. A review of existing literature on QA under the OSSD has been conducted. The hypothesis has been posed that there might be significant potential for formalized user testing under the OSSD if volunteer community members had access to a human computation platform to aggregate their results. The introduction of example use cases for QA in the UBports Community illustrated this hypothesis. This illustration provided grounds for evaluating two existing software solutions to tap into this potential.

Based on these insights, the blueprint for a new software solution - OPEN-CUTS, the open crowdsourced user-testing suite - has been created. The software facilitates QA in open source communities by having inexperienced community members generate formalized user testing data.

An early prototype of OPEN-CUTS has been implemented and evaluated in a usability study. This study's results support the hypothesis that using HC to facilitate QA under the OSSD may be beneficial. The implemented prototype seems to be equipped to tap into this potential. A list of future development goals for OPEN-CUTS has been compiled from user feedback and is provided in appendix 7.2.

6.2 Future Work

In the future, a field study could be conducted to investigate the impact of using OPEN-CUTS in an open source community compared to previously used QA practices. Additionally, a long-term study that examines the adoption of OPEN-CUTS as the primary QA mechanism in a reasonably sized open source community could provide further insights into the following questions:

- What is the technical background of the users using OPEN-CUTS to run tests? Have they contributed before?

- How does the introduction of OPEN-CUTS impact overall contribution behavior? In particular, are other low-skilled areas of contribution (e.g., translations, marketing, user discussion, bug tracker activity) affected?
- What is the contributor retention rate? I.e., how many contributors stay on as active testers, how many merely occasionally contribute test results, how many lose interest?
- Is the introduction of OPEN-CUTS considered helpful by a) the core development team, b) the contributing developers, and c) the bug reporters?

6.3 Conclusion

The need for formalized user-testing and the potential for crowdsourcing in open source communities has been established. A software platform to facilitate this has been designed, and a prototype has been successfully implemented. Fitness-for-use has been demonstrated through a rudimentary usability study. Future research and development will be published on https://www.open-cuts.org.

Chapter 7

Appendix

7.1 Database Schema

7.1.1 User

- `name: String`: the chosen public username
- `email: String`: the email address used to register
- `passwordHash: String`: a salted hash of the user's password
- `confirm: String`: contains the confirmation code if the user has not yet confirmed their email address
- `token: String`: the API token used for authenticating requests
- `groups: [String]`: used for permission management

7.1.2 System

- `name: String`: name of the system
- `description: String`: a brief description
- `image: String`: url[1] of a preview image
- `properties: Array` used to account for different circumstances when testing, eg devices of Ubuntu Touch or packages and host operating systems for the UBports Installer
  - `name: String`: name of the property
  - `description: String`: a brief description of the property
  - `link: String`: an optional documentation link explaining the property
  - `values: String`: possible values of the property that can be specified by the testers
- `logLink: String`: an optional documentation link explaining how to obtain logs for this system
- `combinationFilter: String`: an OCFL statement that can be used to filter combinations of the property values, see section 4.2.6
- `buildDiscovery: [String]`: a list of arguments used to retrieve builds from different sources

[1] Uniform Resource Locator, an address on the internet

7.1.3 Build

- `tag: String`: unique build tag or release name
- `date: Date`: release date, used for sorting and discarding of old builds
- `system: ObjectId`: reference to the corresponding system
- `channel: String`: builds can be organized in channels, see section 3.1.2.1

7.1.4 Test

- `name: String`: name of the test
- `description: String`: brief description
- `steps: String`: detailed step-by-step instructions for testers
- `weight: String`: the importance of the test
- `system: ObjectId`: reference to the corresponding system

7.1.5 Run

- `date: Date`: time and date when the test was run
- `comment: String`: comment left by the user
- `result: String`: PASS if the test was successful, WONKY if it was partially successful, FAIL if it failed
- `spam: Boolean`: true if the run is spam and should not be considered and published, false otherwise
- `flagged: Boolean`: true if the run was flagged as spam by the system and is pending moderation, false otherwise
- `logs: Array`: logs are text files that errors and debugging information are written to automatically during runtime
  - `name: String`: filename
  - `content: String`: logfile
- `combination: Array`: combination of system properties
  - `variable: String`: property name
  - `value: String`: property value
- `test: ObjectId`: reference to the corresponding test
- `build: ObjectId`: reference to the corresponding build
- `user: ObjectId`: reference to the user that reported the run, if it was not reported anonymously

7.2 Future Development Goals

7.2.1 Gamification, Games with a Purpose

The current prototype of OPEN-CUTS does not make use of gamification, even though it might have considerable potential for keeping users engaged in the reporting process. Simple game elements, such as achievements, leaderboards, and daily streaks, could be easily realized in OPEN-CUTS. A karma system has not yet been implemented, as such a feature requires careful planning and evaluation to prevent users from exploiting the system and gaining permissions too quickly.

Depending on the systems tested, there might also be a potential to create external

testing applications that employ game elements. For example, to test the gyroscope in an Ubuntu Touch device, a game could be devised that uses the gyroscope as a control mechanism. The game can then report the result using an API call, which could automatically attach the relevant logfiles.

7.2.2 Automated Reporting

There is also a potential source for real-world testing data from asking users to submit feedback while using the software. For example, the UBports Installer could prompt the user for consent to submit the test run at the click of a button. All the data required (host operating system, package type, build number, logfile) can then be automatically read and submitted to the API by the Installer without requiring any additional user action[2]. This approach will be primarily interesting for such activities that occur rarely and are only performed once a while by every user. For example, while being prompted for feedback after using the Installer is a negligible disturbance, a user would not want to be interrupted multiple times a day during routine activities.

Similarly, OPEN-CUTS could accept generic crash reports that are not associated with a specific test for any system. Existing automation tools like Apport[3] could be adapted to make this possible. Crash runs could be automatically analyzed for similarities and incorporated into the result presentation.

7.2.3 Improved Combination Logic and Property-first Reporting

The combination logic can be augmented to improve the representation of complex systems. For instance, it should be possible for a build to define properties divergent from the system properties. An example of this is the Ubuntu Touch system image server, which specifies every build's supported devices. The build discovery mechanism could detect those properties where they are required.

Similarly, it might be useful to permit tests to specify OCFL statements to control to what combinations they apply. For example, a device without a micro-SD card slot does not need tests for this functionality. The reporting view would then have to record the properties before selecting a test. Currently, a user selects a system, build, and test first, and then selects the properties as part of the run. To better highlight missing tests and calculate more precise coverage results, it would make sense to move the property selection before the build selection. It would then be useful to allow users to specify a list of property combinations they might regularly use to accommodate the accompanying increased importance of combination selection.

The combination logic could also serve to better highlight which tests failed when and for what combination. OPEN-CUTS could produce a list of minimal combinations that failed or succeeded and display more useful information to make identifying regressions in edge cases easier.

[2] A proof-of-concept implementation is available here: https://github.com/ubports/ubports-installer/releases/tag/0.5.4-beta

[3] https://wiki.ubuntu.com/Apport

7.2.4 Integration into CI/CD Pipelines and Bug Trackers

Thanks to the OPEN-CUTS API, it is already possible to integrate OPEN-CUTS test results into parts of the CI/CD pipeline by having the CI system query whether a build has reached coverage and passed. For example, the Ubuntu Touch release routine could query the latest devel build's test results before promoting a new release candidate. A list of OCFL expressions could specify the exact prerequisites for build promotion.

It is also worth thinking about tracking automated test results in OPEN-CUTS, thus acting as a test case management system. OPEN-CUTS could trigger external automated testing software to run tests and incorporate the results into the presentation.

OPEN-CUTS could also integrate with prevalent issue tracking systems, such as GitHub, GitLab, Launchpad, Bugzilla, and Jira. Failing tests could be linked to bug reports so that OPEN-CUTS can prioritize validating failing tests linked to issues across subsequent builds and update those issues if necessary.

7.2.5 Interdependent Tests

When writing detailed tests for a system, there will be cases where it will be useful to allow one test to depend on another. For example, if an Ubuntu Touch device does not boot, the other tests are rendered moot. OPEN-CUTS should be able to respond to such situations adequately. This feature requires meticulous planning to prevent circular dependencies that might result in unexpected behavior.

7.2.6 Improved Moderation

The moderation workflow implemented in the prototype of OPEN-CUTS is very rudimentary. A range of useful governance features would be relatively easy to implement, such as a moderation queue and the ability to leave discussion comments on runs. Such comments could also allow users to be tagged and notified via email. With a notification system in place, it might also be possible to allow moderators to request a test run of a specific property combination on a specific build of a system to speed up the generation of useful results.

7.2.7 Version-controlled Configuration

While the web interface makes it easy for admins to modify each system's configurations, there is no peer-review mechanism, editing history, or commenting. To address this, OPEN-CUTS could read system configurations directly from Git repositories[4]. Systems to be tested with OPEN-CUTS could include a file written in a data serialization language such as YAML[5] or JSON into their version control repository. This file is then modified through the standard contribution workflow, which provides a peer-review process and a commented edit history and allows contributing developers to propose changes

[4]This technique is used by some modern continuous integration systems. One example of this in practice is the Jenkins continuous integration system, which can be configured by including a configuration file called `Jenkinsfile` in the repository's root directory: https://www.jenkins.io/doc/book/pipeline/jenkinsfile/

[5]https://yaml.org/

(rather than just authorized admins). OPEN-CUTS could poll this configuration file as part of the housekeeping pipeline to detect any changes.

7.2.8 Modularize internal Libraries

Some of the internal libraries used could be exported to standalone npm packages. One example of this is the mongoose-data-source library (see section 4.2.3), which provides functionality that could be useful to a range of other NodeJS-based software projects using Mongoose with a GraphQL API. The OCFL functionality (see section 4.2.6) could also be modularized to allow using the same code in the web interface and the API server.

7.3 Raw Data

7.3.1 Task Completion Times

id	group	task 1	task 2	task 3	task 4	task 5	task 6	task 7	task 8
1	"usr"	1:50	1:16	6:35	1:09				
2	"usr"	1:48	1:39	7:20	1:40				
3	"usr"	3:01	3:40	8:20	2:05				
4	"usr"	3:30	4:55	9:00	2:40				
5	"usr"	2:48	1:38	6:45	2:23				
6	"dev"	1:20	0:46	6:05	0:39	1:43	0:51	2:27	3:02
7	"dev"	2:40	1:49	7:20	0:55	2:05	1:42	3:33	6:15
8	"dev"	3:35	4:20	8:23	2:30	2:00	2:59	4:20	7:12
9	"dev"	2:28	2:43	4:38	1:40	0:49	2:33	4:15	5:28
mean	**"usr"**	**2:35**	**2:37**	**7:36**	**1:59**				
mean	**"dev"**	**2:30**	**2:24**	**6:36**	**1:26**	**1:39**	**2:01**	**3:38**	**5:29**

7.3.2 Questionnaire Answers

id	group	s 1	s 2	s 3	s 4	s 5	s 6	s 7	s 8	s 9	s 10	s 11
1	"usr"	5	5	5	5	2	5					
2	"usr"	4	5	5	4	1	5					
3	"usr"	4	4	5	3	1	4					
4	"usr"	5	5	5	5	2	5					
5	"usr"	4	5	5	5	1	5					
6	"dev"	5	4	5	3	1	5	5	2	4	1	4
7	"dev"	3	3	4	2	3	5	3	4	5	2	3
8	"dev"	5	4	5	5	2	5	4	1	4	4	2
9	"dev"	4	5	5	4	1	5	5	2	3	1	4
mean	**"usr"**	**4.4**	**4.8**	**5**	**4.4**	**1.4**	**4.8**					
mean	**"dev"**	**4.25**	**4**	**4.75**	**3.5**	**1.75**	**5**	**4.25**	**2.25**	**4**	**2**	**3.25**

7.4 Think-Aloud Comments

7.4.1 General

- The text is nice and big, which makes it easy to read.
- Is there a dark mode for the UI?
- It'd be good to be able to change some administrator settings. Like the number of required runs, for example.
- Markdown or some other formatting would be cool. Like in the run comments, test instructions and so on.
- Maybe the registration email could just have a confirmation link?
- Kinda annoying that i have to put email here, why no Ubuntu SSO [Single-Sign-On (SSO) services facilitate login and user registration using third-party authentication providers]?

7.4.2 Reporting

- Understanding the test results takes a second when you're first doing it, but after that it's much easier. Maybe there could be a tutorial popup?
- Looking for logfiles takes a lot of time.
- I feel like some tests are a little too specific. You have to read a lot of text for these tiny tasks.
- This might be something someone would use from time to time for short sessions, but for longer use it's… It's kind of boring? It's just not engaging enough.
- The test coverage would be more helpful if it were calculated based on the selected property combination. Maybe the properties should be selected before the build and test? Or there should be some easier way to see which combinations are missing.
- It'd be good to have a confirmation button for the test results. I just clicked the PASS button, but I still wanted to edit my comment.
- If someone has a set of combinations they want to test regularly, it'd be cool to be able to select them with one click. Right now it only saves the last one, which is already helpful, but you'll still have to look sometimes.
- It's cool that you quickly run another test on the same build, you can get into a nice rhythm like that. It could be a little more fluent, but it's a good start!
- Sometimes the loading circle takes a couple seconds, that's annoying.

7.4.3 Moderation

- I feel like the moderation stuff could be expanded a little, maybe this is just because this is only a prototype, but there could be more here in general.
- Ah, that's nice, so my logs aren't completely public. So I don't have to worry about leaking sensitive info as much.
- Linking to GitHub or GitLab issues would be nice. Look at Ubuntu Touch with it's devel builds. It often happens when a new issue is introduced, that it stays open for at least a couple days, so the software could have a tester check if the issue is still there and then after the first result just assume that nothing has changed.
- Why can't you click on the combinations? [A bug was discovered here that combination details do not open if there is only one test, such as the UBports Installer in the samle data.]

- Maybe the software could depict it [regressions in combination axes] a little better, highlight the failing combinations, and so on.
- Maybe it could also be an option to allow editing the runs later?
- If there's a lot of runs, identifying invalid or conflicting results might take a lot of time. Maybe there should be a view where you can see more of the runs and triage them in batches. Like, you don't know if a run is invalid without seeing other runs for the same combination.
- I'd like to be able to comment here, instead of just deleting it.

7.4.4 OCFL

- The combination filtering language is nifty!
- I guess you'll only have to touch this once in most cases, right? Maybe there should be an option to comment it? Complex statements might be more maintainable that way… Like, the installer statement after the assignment, that's already pretty complex if you have to look at it without any prior knowledge.
- I like that you can format it [the OCFL code], but syntax highlighting and auto-complete is missing.
- Matching-Bracket-Highlighting would be poggers [sic!].

7.4.5 Build Discovery

- One-click configurations would be amazing, like importing apps from the build discovery sources directly.
- Editing the build discovery should immediately fetch the new builds, so I can start testing right away.
- There should be some feedback about errors [when specifying build discovery settings]. How do I know if I have any typos?
- The variations should also come from build discovery. Like in Ubuntu Touch, you can have different devices on different channels.
- I'd prefer some more documentation or a better description here [in the build discovery settings]. I had to cheat off of 8192 to find out what to put as an option.

7.4.6 Bugs

- The release dates for the openstore are incorrect.
- This is weird, I can specify property combinations here that are invalid. It doesn't seem to count it to the coverage, but the report is still accepted.

Bibliography

[1] A. M. Turing, "Computing Machinery and Intelligence," *Mind*, no. 49, pp. 433–460, 1950, Accessed: Jan. 22, 2022. Available: http://www.jstor.org/stable/2251299

[2] S. Kumar, "Fundamental Limits to Moore's Law," *arXiv*, Nov. 2015, Accessed: Jan. 22, 2022. Available: http://arxiv.org/abs/1511.05956

[3] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997. Accessed: Jan. 22, 2022. Available: https://www.cs.cmu.edu/~tom/mlbook.html

[4] E. Law and L. von Ahn, *Human computation*. Morgan & Claypool, 2011. doi: 10.2200/S00371ED1V01Y201107AIM013.

[5] R. Mühlhoff, "Human-aided artificial intelligence: Or, how to run large computations in human brains? Toward a media sociology of machine learning:" *New Media & Society*, pp. 1868–1884, Nov. 2019, doi: 10.1177/1461444819885334.

[6] A. Doan, R. Ramakrishnan, and A. Y. Halevy, "Crowdsourcing systems on the World-Wide Web," *Communications of the ACM*, vol. 54, no. 4, pp. 86–96, Apr. 2011, doi: 10.1145/1924421.1924442.

[7] R. Ardila *et al.*, "Common Voice: A Massively-Multilingual Speech Corpus," *arXiv*, Mar. 2020, Accessed: Jan. 22, 2022. Available: http://arxiv.org/abs/1912.06670

[8] K. Huotari and J. Hamari, "Defining gamification: A service marketing perspective," in *Proceeding of the 16th International Academic MindTrek Conference*, 2012, p. 17. doi: 10.1145/2393132.2393137.

[9] L. von Ahn and L. Dabbish, "Labeling images with a computer game," in *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04*, 2004, pp. 319–326. doi: 10.1145/985692.985733.

[10] R. Mühlhoff, "Big Data Is Watching You Digitale Entmündigung am Beispiel von Facebook und Google," in *Affekt Macht Netz*, R. Mühlhoff, A. Breljak, and J. Slaby, Eds. Bielefeld: transcript Verlag, 2019, pp. 81–106. doi: 10.14361/9783839444399-004.

[11] G. G. Schulmeyer, Ed., *Handbook of software quality assurance*, 4th ed. Boston: Artech House, 2008.

[12] S. S. Bahamdain, "Open Source Software (OSS) Quality Assurance: A Survey Paper," *Procedia Computer Science*, vol. 56, pp. 459–464, Jan. 2015, doi: 10.1016/j.procs.2015.07.236.

[13] M. Aberdour, "Achieving Quality in Open-Source Software," *IEEE Software*, vol. 24, no. 1, pp. 58–64, Jan. 2007, doi: 10.1109/MS.2007.2.

[14] E. S. Raymond, “The cathedral and the bazaar,” *First Monday*, vol. 3, no. 2, Mar. 1998, doi: 10.5210/fm.v3i2.578.

[15] D. Wahyudin, A. Schatten, D. Winkler, and S. Biffl, “Aspects of Software Quality Assurance in Open Source Software Projects: Two Case Studies from Apache Project,” Aug. 2007, pp. 229–236. doi: 10.1109/EUROMICRO.2007.19.

[16] D. Kulesz and I. Bogicevic, “SystemTestPortal - A Web-Application for managing Manual System Tests,” Nov. 2017. Accessed: Jan. 22, 2022. Available: https://archive.fosdem.org/2018/schedule/event/systemtestportal/attachments/paper/1969/export/events/attachments/systemtestportal/paper/1969/iste_fosdem2018_v2.pdf

[17] N. Haines, “The Future of Ubuntu,” in *Beginning Ubuntu for Windows and Mac Users*, Berkeley, CA: Apress, 2015, pp. 205–209. doi: 10.1007/978-1-4842-0608-9_8.

[18] B. R. Prathap, “Operating Systems for Mobiles until late 2018 A Comparative Survey and Overview,” *International Journal of Computer Sciences and Engineering*, vol. 6, no. 12, pp. 576–581, Dec. 2018, doi: 10.26438/ijcse/v6i12.576581.

[19] J. Jenkins, “Ubuntu Touch beats Firefox OS to win best of MWC from CNET,” *CNET*, Feb. 2013, Accessed: Jan. 22, 2022. Available: https://www.cnet.com/news/ubuntu-touch-beats-firefox-os-to-win-best-of-mwc-from-cnet/

[20] M. Shuttleworth, “Growing Ubuntu for cloud and IoT, rather than phone and convergence.” Apr. 2017. Accessed: Jan. 22, 2022. Available: https://ubuntu.com/blog/growing-ubuntu-for-cloud-and-iot-rather-than-phone-and-convergence

[21] J. Sprinz, “One year after the world ended - Ubuntu Touch today.” Ubucon Europe 2018, Gijón, 2018. Accessed: Jan. 22, 2022. Available: https://spri.nz/talks/2018/ubucon-europe/

[22] M. Gripsgård and J. Bots, “Stiftungsgeschäft (unter Lebenden) über die Errichtung der UBports Foundation.” Senatsverwaltung für Justiz, Verbraucherschutz und Antidiskriminierung des Landes Berlin, Abteilung II, Stiftungsaufsicht, May 2019.

[23] J. Sprinz, “State of the Touch: Ubuntu on phones and tablets.” Ubucon Europe 2019, Sintra, 2019. Accessed: Jan. 22, 2022. Available: https://spri.nz/talks/2019/ubucon-europe/

[24] J. Sprinz, “Ubuntu Touch & Co - GNU/Linux in der Hosentasche.” 36th Chaos Communication Congress, Leipzig, 2019. Accessed: Jan. 22, 2022. Available: https://spri.nz/talks/2019/36c3-oio/

[25] UBports Foundation, “Release Schedule,” *UBports Documentation*. 2019. Accessed: Jan. 22, 2022. Available: http://docs.ubports.com/en/latest/about/process/release-schedule.html

[26] UBports Foundation, “Contributing: Quality Assurance,” *UBports Documentation*. 2019. Accessed: Jan. 22, 2022. Available: http://docs.ubports.com/en/latest/contribute/quality-assurance.html

[27] C. Győrödi, R. Győrödi, G. Pecherle, and A. Olah, “A comparative study: MongoDB vs. MySQL,” in *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, Jun. 2015, pp. 1–6. doi: 10.1109/EMES.2015.7158433.

[28] Ecma International, “The JSON Data Interchange Syntax Standard,” Ecma International, Geneva, ECMA-404 2nd Edition, Dec. 2017. Accessed: Jan. 22, 2022.

Available: https://www.ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf

[29] B. Terlson, B. Farias, and J. Harband, "ECMAScript® 2019 Language Specification," Ecma International, Geneva, ECMA-262 10th edition, Jun. 2019. Accessed: Jan. 22, 2022. Available: https://262.ecma-international.org/10.0/

[30] W3C, "Web Services Architecture," World Wide Web Consortium, Working Group Note 11, Feb. 2004. Accessed: Jan. 22, 2022. Available: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/wsa.pdf

[31] M. Bryant, "GraphQL for archival metadata: An overview of the EHRI GraphQL API," in *2017 IEEE International Conference on Big Data (Big Data)*, Dec. 2017, pp. 2225–2230. doi: 10.1109/BigData.2017.8258173.

[32] Facebook, Inc., "GraphQL Specification: June 2018 Edition," Facebook, Inc., Jun. 2018. Accessed: Jan. 22, 2022. Available: http://spec.graphql.org/June2018/

[33] D. Michie, "'Memo' Functions and Machine Learning," *Nature*, vol. 218, no. 5136, pp. 19–22, Apr. 1968, doi: 10.1038/218019a0.

[34] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. New York, NY: Springer New York, 2009. doi: 10.1007/978-0-387-84858-7.

[35] MongoDB, Inc., "Aggregation Pipeline," *MongoDB Manual*. 2021. Accessed: Jan. 22, 2022. Available: https://docs.mongodb.com/manual/core/aggregation-pipeline

[36] MDN contributors, "Introduction to client-side frameworks," *MDN Web Docs*. Nov. 2021. Accessed: Jan. 22, 2022. Available: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction

[37] C. M. Lonvick and T. Ylonen, "The Secure Shell (SSH) Connection Protocol," Internet Engineering Task Force, Request for Comments RFC 4254, Jan. 2006. doi: 10.17487/RFC4254.

[38] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Internet Engineering Task Force, Request for Comments RFC 8446, Aug. 2018. doi: 10.17487/RFC8446.

[39] J. H. Birns, K. A. Joffre, J. F. Leclerc, and C. A. Paulsen, "Getting the Whole Picture: Collecting Usability Data Using Two Methods - Concurrent Think Aloud and Retrospective Probing," *Proceedings of UPA Conference*, pp. 8–12, 2002, Accessed: Jan. 22, 2022. Available: https://www.researchgate.net/publication/228865154_Getting_the_Whole_Picture_Collecting_Usability_Data_Using_Two_Methods--Concurrent_Think_Aloud_and_Retrospective_Probing

[40] S. Jamieson, "Likert Scales: How to (ab) Use Them," *Medical education*, vol. 38, pp. 1217–1218, Jan. 2005, doi: 10.1111/j.1365-2929.2004.02012.x.