

THE UNIVERSITY OF CALGARY

Build Notifications for Agile Software Development Teams

by

Ruth Margaret Ablett

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

SEPTEMBER - 2007

© Ruth Margaret Ablett 2007

UNIVERSITY OF CALGARY
FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Build Notifications for Agile Software Development Teams" submitted by Ruth Margaret Ablett in partial fulfillment of the requirements of the degree of Master of Science.

Supervisor, Dr. Ehud Sharlin, Department of Computer Science

Supervisor, Dr. Frank Oliver Maurer, Department of Computer Science

Dr. Jörg Denzinger, Department of Computer Science

Mr. Ron Murch, Haskayne School of Business

Date

Abstract

Continuous integration is an important part of agile software development. This thesis describes the development and testing of a robotic interface developed to assist with the continuous integration process utilized by agile software development teams. This robotic notification device is compared to both ambient and virtual notification mechanisms to evaluate its feasibility in an agile environment. The results of the evaluation suggest that introducing such systems into an Agile environment should be undertaken with care and consideration for the team culture and members' preferences.

Acknowledgements

Thank you to my supervisors, Frank Maurer and Ehud Sharlin, who helped me tremendously over these last two years. I'm so glad I made the choice to come back to Canada. You've helped me to achieve what I didn't think I would ever do.

I would also like to thank all members of the research groups over the past two years: Chengyao (Cupcake) Deng, David Fox, Cheng Guo, Robert Morgan, Xueling Shu, Cody Watts, Patrick Wilson, Min Xin, and Jim Young. I also want to thank the members of the LSMR research group for making our lab such a fun place to be! Much appreciation also goes to the Natural Language Research Lab at Hokkaido University, and to Dr. Kenshi Araki and Dr. Rafal Rzepka for giving me the opportunity to spend a wonderful summer doing research there.

Much appreciation also goes to my parents and family for your help in supporting me and proofreading my work. Thanks to Craig Schock for the many hours of proofreading my papers and this thesis. Thanks also to all the members of the Midnight Taiko Kai, for your friendship and the opportunity to create wonderful music together.

Dedication

To my family

Table of Contents

Approval Page	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
Table of Contents	vi
List of Figures	x
List of Abbreviations	vii
Publications	viii
CHAPTER ONE: INTRODUCTION	1
1.1 Research Goals	2
1.2 Structure of this Thesis	3
CHAPTER TWO: RELATED WORK	4
2.1 Related Work in Software Engineering	4
2.1.1 Software Testing	5
2.1.2 Continuous Integration	7
2.1.3 Knowledge Sharing	8
2.1.4 Build Status Awareness Dispersion	9
2.1.4.1 Methods of Information Signaling	9
2.1.4.2 Current Techniques	12
2.1.4.2.1 Virtual	12
2.1.4.2.2 Environmental / Ambient	15
2.1.4.3 Build notification techniques used in industry	19
2.2 Related Work in Human-Robot Interaction	20
2.2.1 Robots as colleagues and helpers	21
2.2.2 The role of presence	22
CHAPTER THREE: BUILDBOT REQUIREMENTS	24
3.1 Approach	24
3.1.1 A Robot as an Information Radiator	25
3.1.2 Hardware Requirements	27
CHAPTER FOUR: BUILDBOT IMPLEMENTATION	29
4.1 Architecture	29
4.2 The Robot	31

4.2.1 Hardware	31
4.2.2 Robotic Behaviour	32
4.2.3 Low-Level Implementation	33
4.2.3.1 Vision	34
4.2.3.2 Localization and Navigation	37
4.2.3.3 Other components	40
CHAPTER FIVE: FIRST EVALUATION	41
5.1 Context and Approach	41
5.1.1 Participants	42
5.1.2 Scheduling	42
5.1.3 Location	42
5.1.4 Study Implementation	43
5.2 Results and Discussion	45
CHAPTER SIX: SECOND EVALUATION	50
6.1 Study Objectives	50
6.2 Study Description	50
6.2.1 Participants	51
6.2.2 Scheduling	52
6.2.3 Location	53
6.2.4 Implementation	55
6.2.4.1 Guidelines for Developer Participants	56
6.2.4.2 Java Lava Lamps Implementation	58
6.3 Evaluation Results	60
6.3.1 Results for Part I: Email Only	60
6.3.2 Results for Part II: Email and <i>Java Lava Lamps</i>	62
6.3.3 Results for Part III: Email and BuildBot	67
6.3.4 Final Results: Comparing All Three	71
6.3.4.1 Positive and Negative Aspects of Email	74
6.3.4.2 Positive and Negative Aspects of <i>Java Lava Lamps</i>	76
6.3.4.3 Positive and Negative Aspects of BuildBot	78
CHAPTER SEVEN: DISCUSSION	82
7.1 Development Environment Culture	82
7.1.1 What happened to the commit token?	82
7.1.2 Should the development environment be fun?	83
7.2 Notification Device Alternatives	83

CHAPTER EIGHT: CONCLUSION AND FUTURE WORK	85
8.1 Conclusion	86
8.2 Future Work	87
References and Citations	89
Appendix A: First Evaluation Post-Questionnaire	93
Appendix B: Evaluation Pre-Questionnaire	95
Appendix C: Evaluation Explanation and Guidelines - Part I	97
Appendix D: Evaluation Post-Questionnaire – Part I (Developer)	98
Appendix E: Evaluation Post-Questionnaire – Part I (Observer)	100
Appendix F: Evaluation Explanation and Guidelines - Part II	101
Appendix G: Evaluation Post-Questionnaire – Part II (Developer)	102
Appendix H: Evaluation Post-Questionnaire – Part II (Observer)	105
Appendix I: Evaluation Explanation and Guidelines - Part III	107
Appendix J: Evaluation Post-Questionnaire – Part III (Developer)	108
Appendix K: Evaluation Post-Questionnaire – Part III (Observer)	111
Appendix L: Participant Consent Form	113
Appendix M: Ethics and Co-Author Approval	116

List of Figures

Figure 2.1: Ambient Orb	11
Figure 2.2: Dangling String	11
Figure 2.3: A sample build management server web page	13
Figure 2.4: A screenshot of an email sent detailing the build status	14
Figure 2.5: <i>Java Lava Lamps</i> shown in various states	16
Figure 3.1: BuildBot arriving at a developer's desk	27
Figure 4.1: Overview of the BuildBot system architecture	30
Figure 4.2: A straight line, a curved line, and a junction	33
Figure 4.3: Examples of lines on different floors.	35
Figure 4.4: An example of conversion	36
Figure 4.5: An example of BuildBot's internal map	37
Figure 5.1: The laboratory layout for the first evaluation	43
Figure 5.2: The post-questionnaire results from both days of the first evaluation	48
Figure 6.1: The layout of the laboratory	54
Figure 6.2: Line of sight from different developer workstations	55
Figure 6.3a: A developer working with the commit token	57
Figure 6.3b: The commit token itself	57
Figure 6.4: Setup for <i>Java Lava Lamps</i>	59
Figure 6.5: Part I Results - Notification	62
Figure 6.6: Part II Results - Notification – Developers	63
Figure 6.7: Part II Results - Notification – Observers	63
Figure 6.8: Part II Results: View of the <i>Java Lava Lamps</i> from workstation and build awareness	64

Figure 6.9: Part II Results – Perception of Surroundings	65
Figure 6.10: Part II Results – Perceptions of <i>Java Lava Lamps</i>	66
Figure 6.11: Part II Results – <i>Java Lava Lamps</i> ’ audio and visual distraction	67
Figure 6.12: Part II Results – Build Awareness	68
Figure 6.13: Part III Results – Perception of Surroundings	69
Figure 6.14: Part III Results – BuildBot – audio and visual distraction	70
Figure 6.15: Part III Results - Perceptions of BuildBot	70
Figure 6.16: Part III Results: Developers	71
Figure 6.17: Part III Results: Observers	72
Figure 6.18: Part III Results – Preferred notification device - Developers	73
Figure 6.19: Part III Results – Preferred notification device - observers	73

List of Acronyms and Abbreviations

AIBO – Artificial Intelligence roBOt

LAN – Local Area Network

LED – Light Emitting Diode

RSS – Rich Site Summary

TDD – Test Driven Development

Publications

Materials, ideas, and figures from this thesis have, in part, appeared previously in the following publication:

©2007 IEEE. Reprinted with permission, from R. Ablett, E. Sharlin, F. Maurer and J. Denzinger, and C. Schock, "BuildBot: A Robotic Self-Supervision Mechanism for Agile Software Engineering Teams", IEEE RO-MAN 2007, Jeju Island, Republic of Korea, IEEE Press.

Chapter One. Introduction

When writing software, there are many factors that can cause changes in a software development plan. Changes in business environments, customer tastes, budget and time constraints are all inherently part of the software development process. Traditional software engineering involves a heavy emphasis on design and specification at the beginning and then relatively little feedback from customers or users during implementation. Agile methods (Manifesto, 2007) are a set of best practices that allow the customer to take an active role throughout the development process. Instead of one large project over many months or years, the development is broken into many small sub-projects, or iterations. Each iteration is one to six weeks in length (Larman, 2004), and includes analysis, design, implementation, testing, and refactoring. After each iteration, the development team reviews their progress with the customer, who then advises how development should proceed.

The software produced by the developers must not only fit with the customer's vision of the system, but it should be thoroughly tested and working properly. When working in a software development team, it is imperative that the code newly written by each member of the team integrate properly with the code already written. The practice of automatically incorporating new code into the repository, and deploying and testing the result is called *continuous integration*. When the new code is integrated, there may be conflicts with other parts written by the other developers on the team. If these conflicts occur but the team is not aware of them, they may be compounded, adding extra person-

hours to the project. Thus, if these errors do occur, it is very important that the developers be alerted to any such problems as soon as possible so they can be fixed immediately.

Mechanisms to notify developers of such problems do exist and are used in agile environments in industry. Examples include sirens, coloured lights, a message to a shared pager, or virtual notifications such as email, but to the author's knowledge, there has been no empirical exploration of the effectiveness of different types of these devices in a formal evaluation. In this thesis I attempt to explore whether a physically present notification device, in the form of a robot, affects the developers in a positive way.

The tool developed was BuildBot, a robotic build notification device. A robot has an interesting duality – it is a physical entity, present in the development space, but it is also part of the virtual environment of computers. It was hoped that taking advantage of this dual nature would result in a more effective notification device. This robot was tested against two other conventional notification mechanisms in a study with an agile team.

1.1 Research Goals

The overall goal of this thesis is the following:

Investigate the use of a robot as a continuous integration build notification mechanism, and compare this robot with existing virtual and ambient notification mechanisms.

Two evaluations were performed, comparing three different mechanisms: a totally virtual notification, an ambient device, and a robot.

1.2 Structure of the Thesis

This thesis takes the following form: Following this Introduction, Chapter Two provides background and previous work, both in agile software engineering and in human-computer and human-robot interaction. Chapter Three outlines the design goals of BuildBot and Chapter Four describes its implementation in detail. Chapter Five provides the context, implementation and results of the first evaluation, which measured reaction times and participant distraction of unrelated alerts. Chapter Six outlines the context, implementation and results of the second evaluation, which measured the longer-term effects of the build notification. Chapter Seven summarizes the results and experiences of the participants in the study, and Chapter Eight concludes this thesis, outlining shortcomings of BuildBot and the study, and provides suggestions for future work.

Chapter Two. Related Work

The goal of BuildBot was to develop a prototype of a near-autonomous robotic build notification device, designed and tuned for a focused agile software development social setting. At the time of writing, there was no research being done exploring the possible roles of robots in an agile software development environment. However, there is relevant agile software engineering work done that directly relates to this project. The first part of Chapter Two provides a background for the reader, including describing background concepts important to the full comprehension of this thesis. Previous work in agile software development follows, including a discussion of current techniques for build state awareness. The final part of the chapter contains previous work in human-robot interaction.

2.1 Related Work in Software Engineering

Agile methods (such as eXtreme Programming [Beck, 2000] or Scrum [Rising et al, 2000]) refer to human-centric software engineering methodologies that advocate the development of high-quality software using short iterations. The goal of software engineering is to create software that fulfills the needs and wishes of the customer. Software testing is a process used to ensure that the software functions correctly.

2.1.1 Software Testing

Software testing is a way to ensure software functions correctly, safely, and as the customer specified. Manual testing involves a tester running the program and ensuring it produces the correct output. Tests should be run every time a change is made to the code. If an error manifests, it can be found and fixed quickly, as there is only a small amount of changed code that could have caused it. A simple bug which may take only a few minutes to repair immediately after it is introduced into the code base may end up costing significant numbers of person-hours if it is not identified until more code has been written around it.

Manual testing is monotonous and a programmer may miss features or not test features thoroughly enough, especially when having to repeat tests often. The solution is to write tests that will be automatically run using a testing tool such as JUnit (2007), NUnit (2007), Cactus (2007), or some other automated testing framework. Automated testing eliminates the tedious task of manual re-testing so that developers can concentrate on writing code and new automated tests.

Unit testing is the procedure used to finely test individual units of the software (Robillard et al, 2003). Test Driven Development (TDD) is an agile practice in which developers write these unit tests and add them to the test suite *before* they write the code. Doing so sets precise goals for the programmers as it defines the code that needs to be written, and the developer must think about the behaviour of a method before beginning

to write it. It also gives a small sense of accomplishment when tests pass. Most importantly, following this practice ensures tests actually *get* written, instead of the testing period being omitted due to time constraints (Larman, 2004 and Astels, 2003).

An exhaustive suite of unit tests should be maintained – for anything that could possibly break. This suite of tests acts as a specifications document, allowing other developers adding to the codebase to understand how to use the components based on the tests. If a task does not have an exhaustive suite of tests, it is not considered completed and cannot be put into production (Astels, 2003).

A test consists of the creation of an initial state and the calling of the method with input parameters and expected output. These are defined by the developer as they consider the correct behaviour of the class. If the output from the software matches the expected answer, the test passes. Otherwise, it fails. Human inspection of test results is unnecessary, as tests can only pass or fail (Larman, 2004).

There is more than one type of unit test failure. If a failed test has never passed, indicating it is a new test – this may not count as a proper failure, as it indicates the developer has not written any code to make it pass yet. Conversely, if a failed test has passed before, it is a regression failure and indicates that new bugs were introduced into the system to make the test fail. Finally, exceptions may occur, indicating possible integration or compilation problems (Astels et al, 2002; Deng et al, 2007; Astels, 2003).

2.1.2 Continuous Integration

Agile methods rely heavily on automated regression testing to ensure internal software quality. *Continuous integration* is an approach used by agile teams working on a common piece of software. Every time developers check new code into the shared source code repository, the entire software is built, deployed and tested against the suite of automated regression tests. Ideally, check-ins occur frequently. Continuous integration with regression testing and frequent check-ins of tiny increments ensure that developers are aware of the condition of existing functionality with the addition of new code. Continuous integration encourages clean design and prevents problems later on in the development process, since bugs can be caught earlier (Fowler, 2006; Robillard et al, 2003). There are a few continuous integration tools available, such as CruiseControl (2007) and Anthill (2007).

After finding newly checked-in code in a repository, a continuous integration server builds and deploys the software, then executes all tests. If one of these tests fails, the continuous integration server sends an indication to the person(s) who checked-in the code resulting in a state change to build failure. It is the responsibility of the who team member checked in the code to ensure that a reported bug is resolved immediately, for example by reverting to an older version or by fixing the problem in another way. If the bug fix is delayed, other team members might synchronize their code with a broken version of the system – resulting in even more effort required to resolve the problem.

A study by Saff and Ernst (2004) evaluated continuous integration when used by a single developer to ensure new code passed regression and unit tests. They found that continuous integration had a positive effect on the completion of programming tasks. Their study shows that even individual developers benefit from continuous testing of their own code. However, we are interested in ways that different Continuous Integration tools affect agile teams as a whole.

When teams use continuous integration, knowledge of the build state must be dispersed in a way that alerts all developers without becoming a distraction. Agile teams have many means of displaying information for the benefit of developers, customers and managers – these are outlined below.

2.1.3 Knowledge Sharing

Excepting face-to-face communication, *knowledge sharing* is achieved in agile teams through information radiators. These are openly displayed artifacts or other means by which developers can gain knowledge about the state of the project without explicitly seeking it (Cockburn, 2001). Human-computer interaction has a related concept, *ambient data displays*, which present information so that “*one does not even need to be looking at it or near it to take advantage of its peripheral clues.*” (Weiser et al, 1996) These types of displays are often very simple and use colour, sound and motion to convey information. According to Kent Beck (Baker, 2005), “*an interested observer should get a general idea of how the project is going in 15 seconds. He should be able to get more information*

about real or potential problems by looking more closely.” This can be achieved using ambient data displays. The benefits can be seen by outside observers and the development team members. They can quickly assess the state of the project, and this provides encouragement and an incentive to improve.

One example is a pair of lava lamps, green and red. If the green lamp is on, the build is fine. If the red lamp is on, the build is broken and needs to be fixed. The lamps should be situated in a location that is as visible as possible to developers, customers and managers can instantly acquire build status information.

For developers, immediate accessibility of build results makes these results more relevant and gives a sense of satisfaction that the software they are developing is actually being tested. Customers can easily ascertain how their investment is progressing.

2.1.4 Build State Awareness Dispersion

2.1.4.1 Methods of Information Signaling

Cadiz et al (2001, 2003) outline three categories into which awareness strategies generally fall: polling, alerts and peripheral awareness.

Polling involves an information source such that the information is accessible when users demand it. This approach unfortunately has two main limitations, namely, high

cognitive burden (the user has to consciously remember to check the information and process what has changed, taking the user away from other tasks) and unreliable updates (the user may poll infrequently and thus may miss important updates). Thus, a user may miss an important event and only find out about it when she polls.

Alerts are a way of overcoming this, bringing the information to the user when an event occurs. The user has a much lower chance of missing the update; however, she can also be interrupted and distracted. This can take away from a task, which requires intense concentration. The effects of alerts and interruption have been explored in several studies (Cutrell et al, 2001; McFarlane et al, 2001). Pagers and cell phones are a ubiquitous example of alert mechanisms.

Email is a special case as it can either be an alert or a polling mechanism. If the user uses an email notification program (examples include Google Talk or MSN Messenger), a user can be alerted to a new email. If not, the user must poll for email updates.

Peripheral Awareness takes advantage of our subconscious and our ability to absorb information from our environment without having to consciously concentrate on its delivery. Essentially, the goal of peripheral awareness “is to present the information such that it works its way into users’ minds without intentional interruptions” (Cadiz et al, 2001). Essentially, these devices move from the periphery to the focus of a user’s attention smoothly (Weiser et al, 1996). An example of this is the Ambient Orb (*Ambient*, 2007), created by Ambient Devices, which consists of a frosted glass orb that changes

colour depending on the data and programming given to it (Figure 2.1). It could be used to convey information such as the status of a software build (Swanson, 2004; Halvorson, 2004) a reminder to take medication (E-Health, 2006), the latest sports results (red if the home team lost, green if they won) or the direction and strength of a stock quote movement.



Figure 2.1. Ambient Orb

A mix of visual and audio signals can be a very powerful tool for information delivery, as with Jeremijenko's Dangling String (Weiser et al, 1996). The string, an eight foot long piece of plastic spaghetti, is attached to a ceiling-mounted electric motor (Figure 2.2). Data about network traffic is input into the motor - each bit passing along the network is represented as a twitch of the motor. Thus, high network traffic causes the string to whirr along, whereas when the network is not being used, the string may only slightly twitch.



Figure 2.2. Dangling String

There are many more examples of ambient devices. Heiner et al. (1999) created a device consisting of a series of water-filled tubes that can convey text or images using

selective bursts of air bubbles. An office plant that reacts to activity in an environment was created by Bohlen et al. (1998). Dahley et al. (1998) created two types of ambient displays, Water Lamp and Pinwheels, that emulate the beauty and subtlety of nature. Wisnesky et al. created an entire room that brings the user into a rich ambient space (Wisneski et al, 1996; Ishii et al, 1998).

The three techniques discussed above - alerts, polling and peripheral awareness - can be combined in the same context. For example, less important updates can be delivered via peripheral awareness or polling, and something more urgent can demand the user's attention via alert.

There have been a few approaches for how best to disperse awareness information to people involved with a project. These are outlined below.

2.1.4.2 Current Techniques

There are many variations of the following techniques which notify developers or those directly involved on the project.

2.1.4.2.1 Virtual

The first is the simplest approach that works for distributed groups. There may be a central server page (for example, a build status page where those involved with the

project can access the build status from anywhere – see Figure 2.3) which uses the polling approach, or the result may be sent via email (Figure 2.4) whenever a build occurs.

Build Management Server

[Refresh](#)

Anthill Properties

Projects

Project1	Edit	Build	Versioned Build	Delete	19/05/07 6:45 PM
Project2	Edit	Build	Versioned Build	Delete	23/01/07 4:14 PM
Project3	Edit	Build	Versioned Build	Delete	04/05/07 3:08 PM

[Create New Project](#)

Dependency Groups

[Create New Dependency Group](#)

Schedules	Minutes to next build		
15 min	6	Edit	Delete
2 hours	6	Edit	Delete
default	6	Edit	Delete
stoppedSchedule	0	Edit	Delete

[Create New Schedule](#)

Figure 2.3. A sample build management server web page generated by the build engine, in this case Anthill. Anyone authorized to do so can access the page to check the build status. If the build has failed, the user can get more information, including which tests failed.

```
> Date: Fri, 4 May 2007 15:09:54 -0600
> From: anthill@yourcompany.ca
> To: project_manager@yourcompany.ca, developer1@yourcompany.ca, developer2@yourcompany.ca
> Subject: Project3 1.0.111 build failed
>
> Anthill version 1.8.1.303
>
> Retrieving project files: OK
> Incrementing version: OK
> FAILED to run build script. Message: Failed to Build project. Error: 1
> Project Site: http://nana.yourcompany.ca:8080/anthill/projects/project3
>
> Build Log: http://nana.yourcompany.ca:8080/anthill/projects/project3/buildLogs/Project3-1.0.111-build.log
>
> Publish: NOT REQUIRED
> Got Revisions: OK
> Revision 2615 developer2 03/05/07 5:45:54 PM
>
> Modified files:
> Project3_v2.3/config.xml
> Project3_v2.3/src/Project3/distributed/ServerCommunicator.java
> Project3_v2.3/src/Project3/factory/PersisterConnectionInfo.java
>
> Initials work in dialog box, new config file, server communicator takes port no. as argument.
>
>
> -----
> /opt/anthill/publishDir/Project3/buildLots/Project3-1.0.111-build.log
> anthill- nana:
>
> compile-
```

Figure 2.4. A screenshot of an email sent detailing the build status.

An alternative to the polling approach is the use of some kind of virtual alert. This involves the build engine sending to the developers a structured email about the status of the build, or possibly an instant message or system tray alert. There are some variations to the recipients and the frequency of these notifications, but they all occur under the virtual domain. Email can be used either as a polling technique (on its own) or as an alert technique (if the user has an email notification program, e.g. Google Talk or MSN Messenger).

The email may be sent to everyone working on a project (including project managers), or it may only be sent to the last person to upload code which caused one or more tests to fail: the person who effectively caused the build breakage. The email notifies them of the specific tests that failed and the files that were modified since the last stable build, so the developer has enough information to diagnose the problem.

While this approach is simple and requires very little setup, there are inherent drawbacks. Only a small subset of individuals is aware of the build status, and anyone not receiving these emails must acquire the information through polling. Furthermore, this information will immediately become dated as soon as a different person commits new code. The information can only be ascertained by checking the console of the build server itself – if it is accessible to customers and developers.

Additionally, many emails can result in information overload. It would not be wise to send emails to every member of the team, managers, and customers. If sent out every time the build process was run, this information would soon fade into the background and become ignored (essentially, it would turn into a form of spam). This is especially true if code commits are occurring by many developers very often (Smart, 2006).

2.1.4.2.2 Ambient / Environmental

Savoia (2004; 2004; Clark, 2004; Slashdot, 2004; Pragmatic, 2004) has created an ambient feedback device for continuous integration, the *Java Lava Lamp*. There are two

lava lamps involved, one green and one red (Figure 2.5). The green lamp represents a successful build, and the red lamp a build in which one or more tests have failed. Timing is also significant; because the lamps take a few minutes to warm up, the bubbles mean the lamp has been on for at least several minutes. For a green lamp, this is good, however, bubbles in the red lamp indicate a problem.

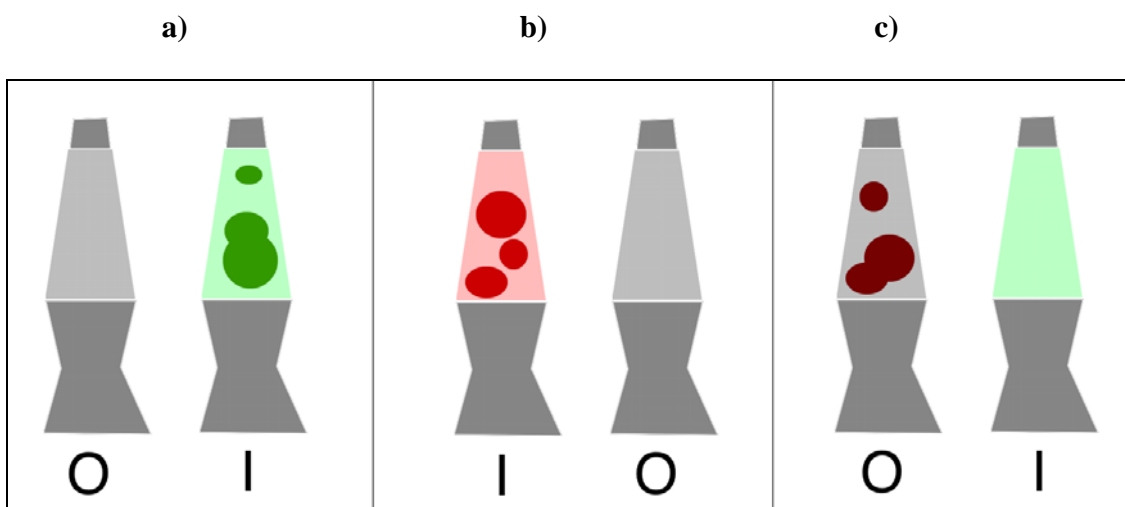


Figure 2.5. *Java Lava Lamps* shown in various states.

- a) The green lamp is bubbling, indicating the build has been successful for at least 10 minutes.**
- b) The red lamp is bubbling, indicating the build has been broken for at least 10 minutes.**
- c) The green lamp is on, but the red lamp's bubbles are still visible, indicating the build was broken for at least 10 minutes but was recently fixed.**

Since lava lamps take a few minutes to heat up, it is possible to tell how long the build has been broken. This delay affords the developers the opportunity to fix a broken build before the lava lamp indicates that the build has been broken for a long time. The playful nature of the lava lamp encourages the developers to fix the build without requiring the involvement of the project manager.

In Clark's implementation of *Java Lava Lamps* (Clark, 2004), there is a clicking sound made when the receiver module switches the power to the lamps on and off. This adds to the soundscape of the developers' area. The click is audible without being obtrusive, so it can alert about a change in the build without becoming a distraction.

The lava lamps are usually combined with email. The person responsible will see the lava lamps change, hear the click, and will receive a notification that their recently committed code broke the build. The build engine's notification should include the failed tests, files which contain offending code and anything else that will help the developer to fix the problem as soon as possible. The lamps themselves give no information other than that the build has broken –only really useful to those who want a high-level overview of the status of the project (such as customers or perhaps managers).

When combined with email, these *Java Lava Lamps* are a vast improvement over email alone, but still have limitations. For example, there is no individual accountability – everyone knows the state of the build, but there is only a slight pressure, for the sake of the group, for the individual to fix their offending code.

Savoia's *Java Lava Lamps* were developed to help a team keep track of the build status through two coloured lamps located in a prominent location in the development area. These lamps are a simple way of conveying the build state, which is accessible via the colour of the lamp currently lit. However, this is the only means of information dispersion. If the build is broken, the lamps keep the developer responsible for the broken

build anonymous. The person whose code caused tests to fail should feel some kind of pressure to fix the build. This pressure should not be unpleasant or cause the developer any more than slight embarrassment. It is hoped that introducing pressure in a fun and lighthearted way will be most effective.

The lava lamps show the build state through the colour of the lamp currently glowing. The lava lamps seem to be successful when people are walking around or entering the work environment, especially if it is made up of cubicles. However, for a developer seated at her workstation to access this information, she must look at the lamps. If she does not have a clear view of the lamps from her workstation, it defeats the purpose of an ambient system for her, as she must expend effort to learn the status of the build. One way around this is to use sound to diffuse information (disadvantages include the potential for distraction, or the fact that developers may be wearing headphones and thus not notice).

When the power is switched from the red lamp to the green lamp, a soft click can be heard. While audible, this click is very quiet, is instantaneous, and is identical whether the build was broken or repaired. A more robust solution would emit some sort of audio indication that the build was fixed or broken, and anyone in the room would be able to instantly become aware of the build state through sound alone.

While this approach is simple, it primarily makes use of visual information and this requires developers to poll the ambient display. A better solution would eliminate the

need for polling and instead utilize information osmosis. In this way, information would become available for developers without causing a distraction.

2.1.4.3 Build notification techniques used in industry

In software development teams that practice agile principles, different types of alerts can be used to notify developers, project managers and others involved with the project.

Smaller, co-located teams can rely on inter-team communication for notification of a build failure. Some larger teams, or teams working together, can rely on either email, a system tray notification or RSS feed alert (Smart, 2006). Several types of alerts can be used, one for failures on the development and testing machine, and a more urgent alert for when tests on the deployment machine failed (for example, a page alert on the company pager). For the most urgent situations, the project manager and directors can visit the development area.

The following section provides, for the reader, a background of fundamental concepts as well as current research in human-robot interaction and human-computer interaction.

2.2 Related Work in Human-Robot Interaction

Robots are simply mobile computers, but the psychological effect they have on humans is remarkable. *Affective interfaces* are defined by Picard (1997) as interfaces which relate to, arise from, or deliberately influence emotions (Scheef et al, 2000).

Scheef et al. (2000) studied how children and adults in different social settings reacted to a teleoperated robotic dog named Sparky. Adults and children had a chance to interact with it and became engaged with it. Thrun (1999) studied museum visitors' reactions to a tour guide robot, Minerva. The robot interacted with many visitors in a spontaneous, short-term manner, utilizing emotional emulation to communicate with them. Visitors were attracted to the robot when it emulated a positive, happy mood.

Sparky and Minerva were tested via short-term, spontaneous interactions with people who had not interacted with them before. There may be differences in the relationships humans have with more permanent robots in an environment, such as a companion, helper, or colleague.

Friedman, Khan and Hagman's study on the human-robotic relationship revealed that a person's mental model of autonomous robots are more anthropomorphic in nature than with convention computer systems (Kiesler et al, 2004; Friedman et al, 2003). Very few people give their computers a name, but the majority of owners of the Sony AIBO robot dog seem to do this. Even though they are aware that AIBO is a piece of hardware

programmed like a computer, participants in the study used words such as “eyes”, “ears”, “brain” or other biological descriptors to describe AIBO’s mechanical parts. Similarly, biological processes were also used to describe behaviour: for example, that AIBO “sleeps” (Friedman et al, 2003). This study raises the possibility that people will react more favourably to a robot with an aesthetically pleasing shape, even if the robot’s role is serious.

Entertainment robots represent only a fraction of the roles robots can play. Industrial robots were among the first robots used in significant numbers. However, these robots had very little interaction with humans and were totally preprogrammed to complete a task in a near-unchanging environment. Creating a robot to complete a task that involves interaction with humans in a constantly changing environment presents a great challenge. Many of us dream of having a robotic maid to do the cleaning and cooking, or robotic cars to drive us safely from one location to another. There is research being done to develop robots that perform useful roles and interact with humans.

2.2.1 Robots as colleagues and helpers

Khan (1998) and Dautenhahn and colleagues (2005) performed studies to examine adults’ perceptions of a robot as a potential helper in the home. Generally, people were more inclined to want a robot to perform tasks such as cleaning, but less inclined for tasks such as child and animal care.

Kidd (2003) discusses different uses for sociable robots. He describes the transition between using robots as tools or simple entertainment devices (such as an industrial robot or a children's toy) to becoming partners that interact with humans. While still limited in their capabilities, emerging robots are designed to interact with their users' sociability, and to integrate in a social setting as peers and teammates (Kidd, 2003; Xin et al, 2006; Breazeal, 2002). His study found that participants reacted quite differently to a human, an animated character, and a robot in several game situations. This can be explored in many contexts, beyond the realm of entertainment.

2.2.2 Robots' physical presence and onscreen agents

There have been several studies that examine the psychological effect of physical presence on users.

Young and his colleagues (2006) created an interface through which remotely located colleagues could express "surrogate presence" through a Sony AIBO robot dog. DiSalvo et al. (2003) developed a telerobotic interface, called the Hug, through which users can express intimate communication and touch.

Yamato et al. (2003) performed a study in which participants were asked to perform a task, assisted by either an on-screen agent (an image of a rabbit) or a robot situated near them (shaped like the on-screen rabbit). People reacted more positively to

the robot when they were dealing with a physical interface rather than an on-screen interface.

Wainer and his colleagues (2006) did a series of studies in which participants were helped with a puzzle by a robot situated in the same room, a robot speaking over a teleconferencing link, and a simulated robot. They found that robot embodiment had a significant effect on the participants' impression of the robot's role. Many participants felt that the robot was more enjoyable, helpful and generally appealing than a simulated, on-screen robot.

By extension, physical presence in an ambient display could have an effect on its impact. "Robots can be viewed as computers with an active presence in the physical world" say Young and his colleagues (2006), and this concept has the potential to be a powerful one when applied to notification mechanisms and ambient displays. This was the basis for BuildBot, a display that actually moves and reacts to virtual stimuli physically in the development environment.

It was hoped that a more interactive system would enhance the way the knowledge is shared without becoming a distraction to the intended audience. BuildBot was developed as an engaging and sociable monitoring technique that was hoped would encourage team members to do the right thing, avoiding uncomfortable and perhaps more threatening approaches such as those requiring the project manager's involvement. BuildBot's design is described in detail in Chapter Three.

Chapter Three. BuildBot Requirements

This first part of this chapter outlines possible improvements to previous notification devices and introduces the idea of a robot in that role. The second part describes the design of BuildBot.

With continuous integration, it is very important that the developers, customers, managers, and anyone else involved in a project be aware of the build status. There are several requirements for the type of interface used.

- It must give build information without someone having to seek it
- It must be noticeable, but not too distracting
- It must be easy to interpret, even with a passing glance

3.1 Approach

The goal of BuildBot was to use the agile team's collective awareness to create an engaging and fun tool that helps the team become aware of and thus fix broken builds as quickly as possible. It was decided that the tool should be one that brings the information to the developers, rather than having the developers take time and mental energy to seek it. The tool should not be too distracting, nor should it fade into the background to the point where the information is not readily available.

3.1.1 A Robot as an Information Radiator

BuildBot was developed to bring build information to the developers. If a developer commits code which causes a build breakage, the robot, which usually resides upon a power station, stands up and follows a network of lines to the responsible developer's workstation.

A robot has the potential to be an effective assistant to an agile team. Robots possess the ability to physically respond to virtual stimuli, bringing awareness information from the digital realm into the physical and vice-versa. Robots are equipped with input sensors and output actuators that can allow them to become a believable physical component of the agile team physical setting. Robotic embodiment of the state of the software build can help an agile team collaborate more effectively, especially if the robot is allowed to physically interact with the team members. In this environment, a BuildBot would act as a *dynamic information radiator*, that changes according to the circumstances, rather than a static radiator, which tend to get ignored (Cockburn, 2001). Information exchange thus occurs in an ambient manner, in the physical context of humans (Xin et al, 2006).

If new code integration and testing is successful, BuildBot stays still and silent. Instead of giving positive feedback (which may distract team members), the robot only alerts developers when there is a problem. If new code integration causes one or more tests to fail, the build is considered broken. In this case, BuildBot walks to the individual

whose code is responsible for test failure and displays to the team that it is not happy with that developer, in a friendly, funny, and playful way (Figure 3.1). The timing of this walk is important – BuildBot’s deliberately slow and dramatic walk serves two purposes. It alerts the team to the broken build via ambient sound and visual cues. By sending an e-mail to the responsible individual, BuildBot alerts him to the failed tests and gives him time to fix the problem. After a few minutes, the dog begins its walk to the workstation. This creates a playful tension as the other team members wonder where the robot is going.

The goal of BuildBot’s design is the following: by giving the responsible individual a lighthearted and friendly reprimand, BuildBot introduces more targeted accountability. It avoids potentially unpleasant confrontations (for example, between a team leader and a developer) and it introduces a mild social incentive to fix the problem without having to involve a superior.



Fig.3.1 BuildBot arriving at a developer's desk.

3.1.2 Hardware Requirements

For BuildBot to be an effective monitor for an agile team, it is necessary to have a robot with several capabilities and features. To physically interact with the team members, the robot needs to be able to walk with stability and be able to handle different surfaces such as carpet or linoleum. It also needs to gain information about its environment through vision. In order to act as an information radiator and provide ambient build information, sound or LED lights (or both) are necessary. To communicate

with the continuous integration server, the robot needs wireless capability. Lastly, the robot must be easily programmable.

BuildBot was designed to be a tool so that teams, customers and management can easily see build status information without having to seek it. The implementation and design of BuildBot is described in Chapter Four.

Chapter Four. BuildBot Implementation

Chapter Four illustrates BuildBot, and outlines the interaction between it and the continuous integration components. The implementation of the robot, including its vision, navigation and other facets of its behaviour are described in detail in the second part of the chapter.

4.1 Architecture

There are several different components involved in the BuildBot system. The architecture and inter-component communication are shown in Figure 4.1 below.

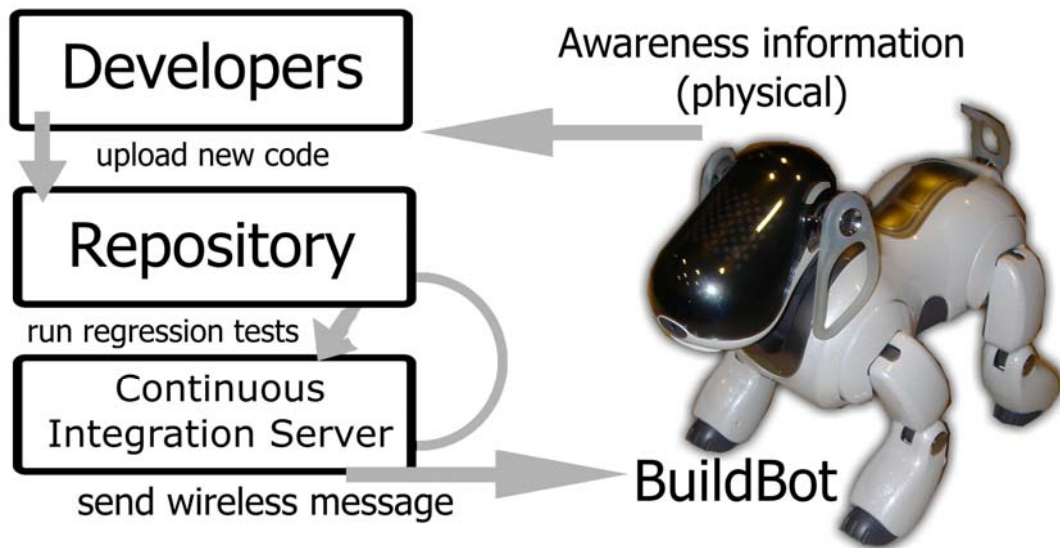


Fig.4.1 Overview of the BuildBot system workflow. The developers commit their code to the repository, which is accessed by the continuous integration server. The continuous integration server compiles, deploys and runs tests and reports the results to BuildBot, which uses ambient awareness information to alert the developers.

The *repository* contains the software currently being developed by the software team. When a change is uploaded to the shared code repository by a team member, the *continuous integration* component runs a script that integrates the entire team's code. The code is compiled, deployed, and a series of tests (this can include unit tests, acceptance tests, and integration tests) is executed. If all tests pass, the result is a successful build. An unsuccessful build can be caused by a compilation error, a deployment error, an exception, illegal operation (such as divide by zero) or one or more failed tests.

The result (a string representing the responsible developer's seat number) is sent to the robot via a wireless LAN. It contains the build status and the office location of the build-breaker. The details of the implementation are explained later in this chapter.

4.2 The Robot

Advances in robotics have resulted in robots that are powerful, affordable and compact enough to make the BuildBot prototype feasible.

The form of the robot used for BuildBot is very important to engage people and avoid unpleasant effects. As robots become more and more lifelike, there is a point where robots stop being cute and engaging and begin to look creepy. Masahiro Mori described this effect as the “Uncanny Valley” (*bukimi no tani*) (Mori, 1970). For example, a humanoid robot with unnatural movements may remind people of a moving corpse! A cute robot dog was thus chosen to avoid this effect and increase the chance that the developers will see the robot as a positive, lighthearted addition to the environment. The robot used for the BuildBot’s implementation is the most recent model of the Sony AIBO robot dog, the ERS-7 (Sony, 2007).

4.2.1 Hardware

The AIBO is ideal for an ambient information radiator as it combines motion, visual, audio, and tactile components. It can move its head and four legs, and comes equipped with a camera, two microphones, a speaker, LEDs in various colours, wireless capability, touch-sensitive buttons, and sensors for temperature, vibration, distance and acceleration. The robot is also zoomorphic, in the shape of a cute puppy, and that adds to the feeling of fun and lightheartedness.

4.2.2 Robotic Behaviour

BuildBot's behavior is event-based. When inactive, the robot stays perched atop its power station, charging, until it receives a message from the server regarding the build status. If the build is successful, BuildBot does not move. However, if the build is broken, BuildBot identifies the responsible developer by getting up and slowly moving to his workstation.

When creating the first prototype of BuildBot, it was necessary to simplify complex details such as the vision. In order to allow the robot to walk to the team member's desk, a vision algorithm was developed that analyzes the streaming video from the robot's eye (a camera integrated into the end of its nose). For simplicity, a tree structure of guidelines was placed on the floor. In this tree, each leaf terminated at a developer's desk and this allowed BuildBot to easily navigate to a specific workstation using the simplest route.

The lines on the floor have junctions which branch at 90 degrees (Figure 4.2). Perpendicular junctions simplify the junction detection algorithm so that BuildBot can easily distinguish junctions from curves.



Fig.4.2. A straight line, a curved line, and a junction.

When walking, the robot tracks junctions and consults an internal map, in the form of a matrix, which gives directions to each workstation. Once BuildBot reaches the end of a line and has passed the correct number of junctions, it has arrived at its goal. BuildBot then looks up and gently reprimands the team member by barking and growling. This robotic reprimand will cease when the build is fixed, or when the robot senses a touch on its head sensor.

4.2.3 Low-Level Implementation

The robot's behavior is written in the C++ programming language, and controlled via the Tekkotsu development framework (Tira-Thompson et al, 2004). Tekkotsu allows developers access to robot and environment information, such as vision and sound streams, temperature, velocity, sensor pressure, joint positions, and battery level. Other information is accessible through the robot's LAN connection. Developers can use this information and write *behaviours*, components which describe the robot's actions in response to events. There are some default events, such as starting or stopping a behaviour, but the developer can define a response to any information to which the robot

has access. For example, if the AIBO ever falls on its back, an orientation-sensing behaviour can ensure it gets back up.

4.2.3.1 Vision

The first step in implementing BuildBot's behavior was to program the robot to follow a single line on the floor. This was done by acquiring the vision stream via Tekkotsu and thresholding the intensity values to create a binary array. The native values of this array lie between 0 and 255, but rarely span that entire range due to lighting conditions. The maximum and minimum values of the entire vision stream are then calculated. Thus, the entire set of values is recalibrated with the maximum being converted to 255 and the minimum converted 0, and all other values lying between the two. By normalising the intensity values, differences in lighting can still result in a readable array as long as the line contrasts sufficiently with the floor. If there is too little contrast (less than 30/255 intensity, before conversion) between the minimum and the maximum, the robot stops – the contrast is assumed to be too low or there is no discernable line. Figure 4.3 contains some examples of flooring types as seen through the robot's camera.



Fig.4.3. Examples of lines on different floors.

- a) The contrast is too low: the robot is off course.**
- b) The line in a dimmer room. The robot can still find the line.**
- c) The line on a variable-colour carpet. The robot can still find the line.**
- d) The line on a lighter carpet. The robot can still find the line.**

Problems can occur if there is an object or flooring pattern in the field of view which affects the minimum and maximum contrasts. In such a case the robot will be unable to find a line but will keep walking, following what it perceives to be the line.

The line detection algorithm is straightforward – every 100 milliseconds, the midpoint of the binary one values (representing the line) in every horizontal row of the binary array is computed (Figure 4.4). The mean of all the nonempty rows is calculated

into a collective midpoint direction. If the resulting midpoint is off center, the robot will rotate itself accordingly in order to compensate. The robot can also follow a curved line, as long as the curve is not too sharp (between approximately -70 and 70 degrees).

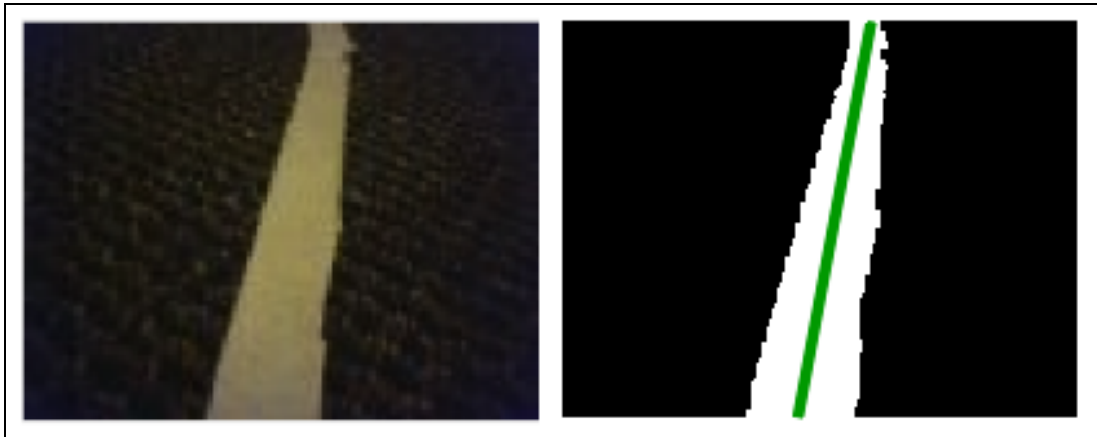


Fig.4.4. An example of conversion. The image is normalized and a threshold is applied to the image. The line direction (shown here in green) is determined by the position and width of the white area of each horizontal row. In this case, the robot would rotate right to keep following the line.

The line colour must be sufficiently contrasting with that of the floor – the best choice of line colour is fluorescent pink. Pink has the highest intensity and is sufficiently rare that there is a reduced chance of the robot mistaking some external brightly-coloured object as the line. Fluorescent pink was not available, so in the test environment, our lines were eggshell white. Despite this, the line tracking algorithm proved to be stable enough, even under the inconsistent and changing light conditions in the laboratory.

4.2.3.2 Localization and Navigation

After implementing the line-following component, the next step was to include support for the detection of junctions. After the line direction has been detected, BuildBot attempts to detect an intersection to the left, right, or in both directions.

The internal map inside BuildBot contains directions on how to get to each team member's desk by following the junctions. For example, in Figure 4.5 below, in order to get to Craig's workstation, BuildBot will turn left at the first junction, go straight at the second, straight at the third, and left at the fourth. A '-1' in the next junction indicates that the preceding non- '-1' value is the last valid junction. The '-1' indicates there are no more junctions; that is, the workstation is at the end of that branch.

```
{LEFT, STRAIGHT, STRAIGHT, STRAIGHT, STRAIGHT, LEFT}, // Rebecca
{LEFT, STRAIGHT, STRAIGHT, STRAIGHT, STRAIGHT, RIGHT}, // Jim
{LEFT, STRAIGHT, STRAIGHT, STRAIGHT, LEFT, -1}, // Michael
{LEFT, STRAIGHT, STRAIGHT, RIGHT, -1, -1}, // Shinya
{LEFT, STRAIGHT, STRAIGHT, LEFT, -1, -1}, // Craig
{LEFT, STRAIGHT, RIGHT, -1, -1, -1}, // Jonathan
{LEFT, STRAIGHT, LEFT, -1, -1, -1}, // Carlos
```

Fig.4.5. An example of BuildBot's internal map.

There are several mechanisms through which the robot can determine that it has become lost. If the robot loses the line or if the lighting is too dim, the robot will check if it has passed the correct number of junctions to be at the correct desk; if not, the robot is assumed to be lost. Also, if the robot detects a type of junction at which its next turn

instruction is not possible (for example, to go left at a right-only junction), it also becomes lost. If the robot encounters a junction whose junction number is higher than the number of junctions (a result of perhaps having wrongly identified a junction where there was none, for example), BuildBot assumes it is lost. When it becomes lost, BuildBot stops and calls for help by playing a quiet whining sound until it is placed back on its power station.

BuildBot moves slowly in order to give the developer enough time to fix the build before arrival at his workstation. In this case, the robot is able to turn around and head back to its starting point, following the line back and keeping track of its location. The algorithms for going to a desk and returning to the power station are almost totally identical (see the following page for pseudocode for the high level junction tracking algorithm), except for that the **junctionCount** variable is decremented rather than incremented. When turning, BuildBot will turn in the opposite way to the direction indicated on its map matrix when returning to the power station.

```

// wait for signal from Continuous Integration server

if (signal == broken && at power station) { // build broken
  if( !seat number exists)
    whine
  else {
    get up from power station
    follow line
    at every junction {
      if(junctionCount == -1 && returning to workstation)
        remount on power station
      else if( junctionMap[signal][junctionCount] == STRAIGHT)
        if(straight possible)
          continue following line
        else
          whine // lost
      else if(junctionMap[signal][junctionCount] == LEFT)
        if(left turn possible)
          turn left
        else
          whine // lost
      else if(junctionMap[signal][junctionCount] == RIGHT)
        if(right turn possible)
          turn right
        else
          whine // lost
    }
    else { // end of the line
      sit down
      growl
      when build signal received
        if(still broken)
          keep growling
        else
          turn around
          follow line // back to power station
    }
    if(going to a workstation)
      junctionCount++
    else // returning from a workstation
      junctionCount--
  }
}

}

at any time, if (power < 30%) { // override all other behaviours
  if(going to a workstation)
    turn around
    follow line back to power station
  else if(returning to workstation)
    follow line back to power station
  else if(growling at developer)
    continue growling
    turn around
    follow line back to power station
}

```

4.2.3.3 Other components

After intersection detection, the major components of the robotic implementation were completed. Aesthetic components, such as posture and barking, were added after the major components were working reasonably well. Power sensing was added as well: if the robot senses its battery power is at 30% or lower, it will return to its power station, even if it was on its way to a workstation.

The continuous integration server sends the build status (and, if broken, the last person to upload code resulting in a state change to a broken build) to BuildBot through the wireless LAN. BuildBot's internal map contains locations in the form of desk numbers. If the build breaks, the message sent is the location number of the developer's desk. If the build is successful, nothing is sent.

BuildBot was designed as a fun form of ambient notification in mind. It was tested alongside two other forms of notification in two separate evaluations. The studies and their results are described in chapters Five and Six.

Chapter Five. First Evaluation

In order to determine developers' reactions to the different notification techniques, two evaluations were conducted. The first evaluation, described in this chapter, describes quantitative results such as average reaction times to the different alerts, as well as participants' feelings toward them. The longer-term second evaluation, in which participants use the alerts in the context of the continuous integration of a project, is detailed in Chapter Six.

The first evaluation is described in detail in the first part of the chapter, and the results presented and analysed in the second.

5.1 Context and Approach

This first evaluation was performed to determine developer's reactions to different notification mechanisms in an informal academic environment. Specifically, the goal was to gauge how effective each device was: how long it took each person to notice each one, and if they noticed it at all.

In this evaluation, the participants were not involved in the same project, and the alerts were sent out randomly. The purpose of this study was to assess the effectiveness of certain notification mechanisms, and to explore the effect each had on the participants.

These notification mechanisms would be potential candidates for use in a development environment for notifying the developers of a build break.

5.1.1 Participants

Two parts of the first evaluation were performed with a peer group of graduate students. The first was done with 4 developers, and the second a group of 10 developers.

5.1.2 Scheduling

This study was performed on two different days. On the first day, the four participants had not seen the robot or the lava lamps in action. On the second day, the ten participants had seen the robot and lamps in action before and thus the novelty effect was reduced.

5.1.3 Location

The study took place in an academic laboratory with the layout shown in Figure 5.1:

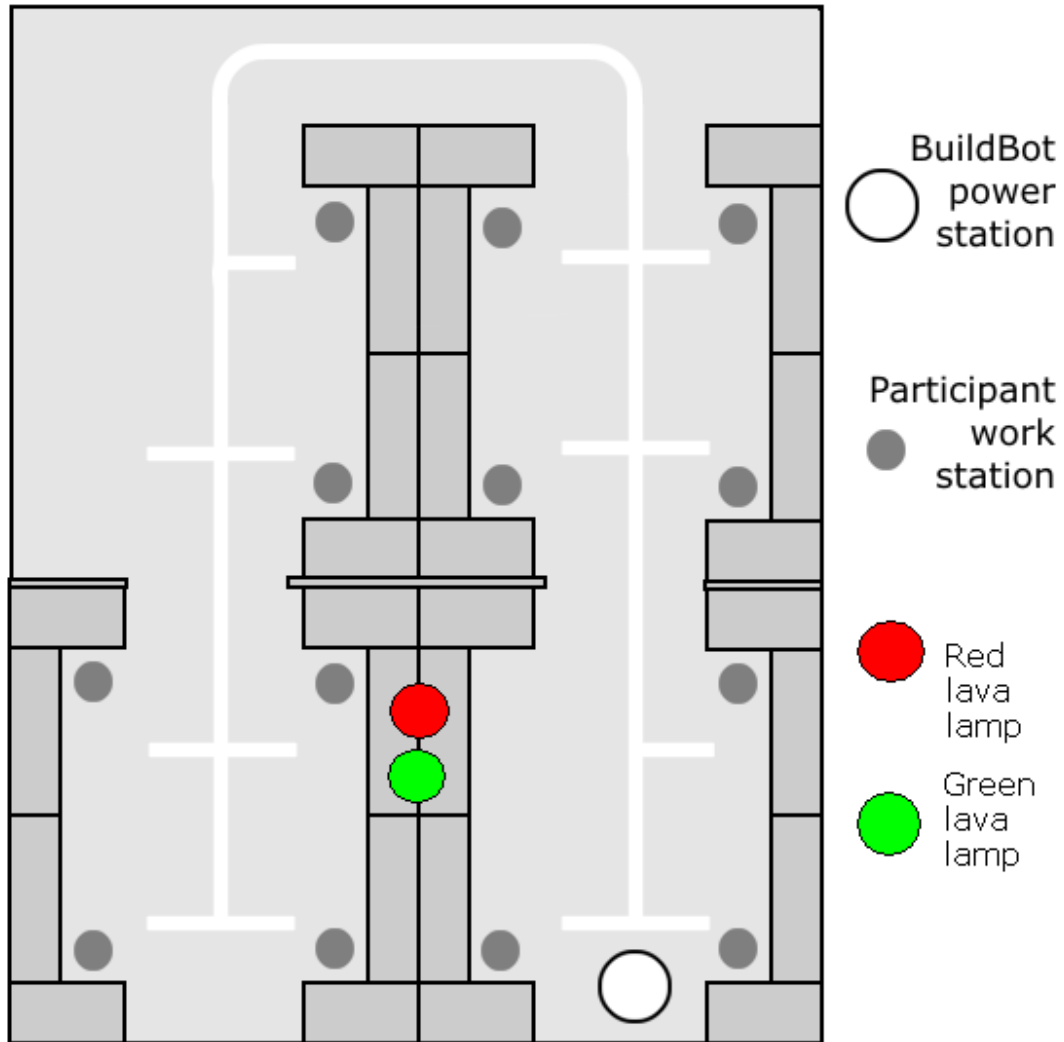


Figure 5.1 The laboratory layout for the first evaluation.

5.1.4 Study Implementation

The participants were allowed to work as they would normally. This includes pair-programming, using an email notification program, wearing headphones and/or listening to music. This altered the chances of some participants noticing the different

notification mechanisms, but was done in order to observe the developers working in their environment.

Three types of notifications were tested in this first evaluation. An entirely virtual notification in the form of an email to all participants, an ambient device in the form of *Java Lava Lamps* accompanied with an email to all participants (see the Previous Work section), and an active alert - the deploying of BuildBot to one of the developers accompanied by an e-mail.

The first notification mechanism, email, was sent with the message “BUILD BROKEN” to all developers. No physical notification of any kind accompanied this email.

The lava lamps were displayed in a prominent position in the development area so that any developer could see it from their desk (some had to turn around to see them, as they were facing the opposite direction). They were switched off or on (switching the illuminated lamp from red to green or vice-versa) via a remote control, and this was combined with an email alerting all developers. The mechanism controlling the lava lamps’ power source makes a clicking sound when the switch occurs, so this also alerted the developers via sound. The third mechanism, BuildBot, was deployed to a developer, selected randomly, and this was accompanied by an email only to that developer.

The case in which the build is successful, where BuildBot does not move from its starting location, was not tested with developers. This is because the absence of the robot's movements is an indication that the build is successful. This study was aimed at determining the effectiveness of different mechanisms – how long it took developers to notice each of the three kinds of notification.

While the participants worked at their workstations on their individual tasks, which were unrelated to each other, a notification of one of the three kinds mentioned above was sent out. The participants were instructed to notify the study facilitator privately, using an instant messaging program when they noticed the robot, received an email, or noticed the lava lamps had changed their state. Sending an instant message ensured that other participants did not become aware of a change through this communication, potentially affecting the results.

The results from this first evaluation are described below.

5.2 Results and Discussion

The results of the first evaluation were as follows: When the email was sent, only the two who had email notification programs noticed within five minutes. In one case, just before the experiment finished, one person checked for email and saw all the email notifications at once.

Generally, when the lava lamps were switched in conjunction with the email, only one person heard the click of the power switch, in this case he did not have a clear view of the lava lamps from his desk. Of the people who responded to the change, half had noticed the lamp had changed, and half had only received the email. The two respondents who responded to the sight of the lava lamps were getting up and walking around the lab when they noticed the lamps had changed.

When BuildBot was deployed, all four participants were alerted by the sight or sound of the robot walking. Three out of the four participants noticed when the robot was near their desks.

The participants did not seem to notice the lava lamps, even though they are in a prominent location, visible to as many people as possible. However, when BuildBot was deployed, the combination of sight and sound resulted in all participants noticing the robot when it was near their desk. As a result of this, every participant agreed that the robot was a distraction. However, every one of the participants also either agreed or strongly agreed that the robot also contributed to a fun atmosphere. We observed that the energy in the room increased noticeably when the robot was walking to a developer's desk – people got up and started talking, walking around, and wondering what was going on.

During the second evaluation, the results were more polar. The lamps were switched six times during the course of the second day of the first evaluation, and one

participant noticed only one of these changes - when this participant returned to the room and found the red lava lamp lit. The robot, on the other hand, was sent out twice, and both times, eight out of the ten respondents noticed it. The respondents who did not notice the robot were out of the room when it ran.

During both days, the developers nearest to BuildBot noticed it when it was near them, even if it was on the other side of a cubicle wall. On the first day, it took participants a mean of 5.1 minutes to notice BuildBot. On the second day, it took participants a mean of 3.3 minutes.

For some participants, especially those listening to music or talking to other people during the study, they did not notice the robot until it was near their workstation. Some participants were more distracted by the robot's movements because their desks face outwards, towards aiseways the robot used to arrive at different desks. It was observed that participants who did not find the robot's movements distracting had desks that faced the walls of the lab and not the aiseways where the robot was likely to be present.

Figure 5.2 shows the data collected from the post-study questionnaire. This questionnaire is listed under Appendix A.

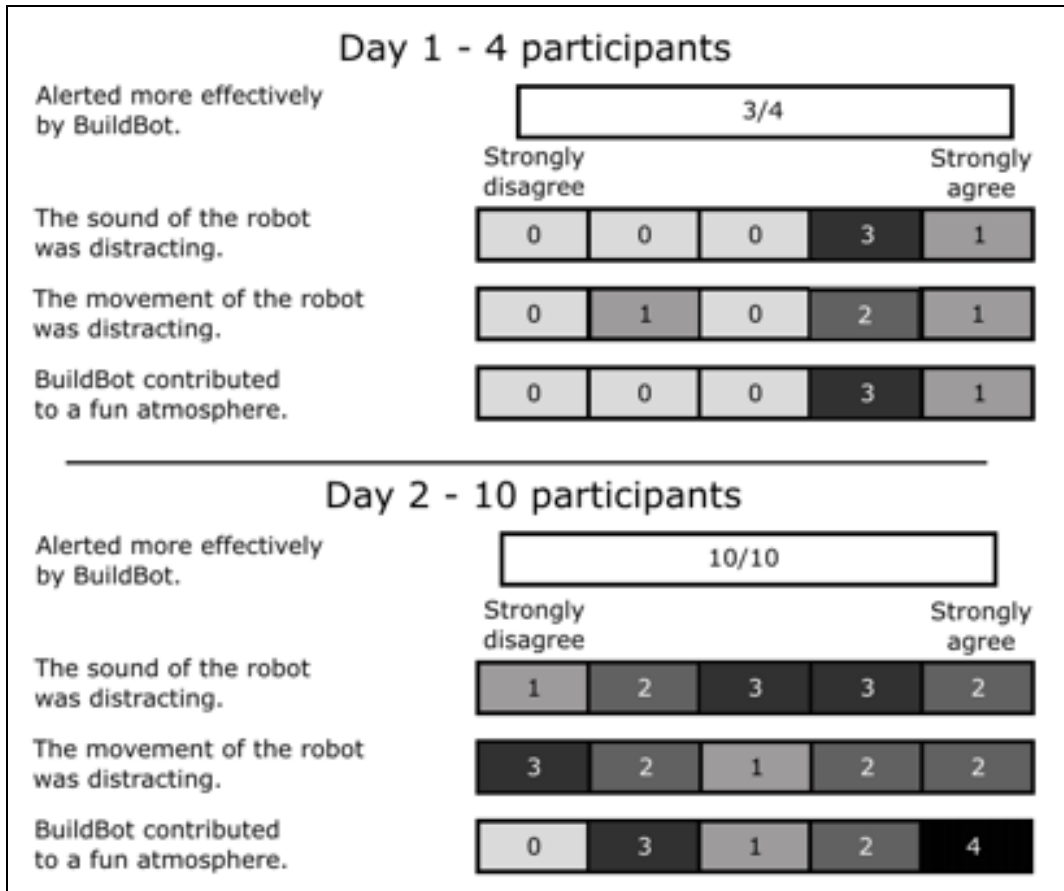


Figure 5.2 The post-questionnaire results from both days of the first evaluation.

Developers were easily alerted by the movement of BuildBot, but found it somewhat distracting when the robot was in motion. Despite this, most developers felt that the robot contributed to a fun atmosphere.

From these results, we can see that generally, BuildBot was more effective at alerting developers when a change occurred. Although it was somewhat distracting, most developers felt that BuildBot contributed to a feeling of fun.

The second evaluation, in which the alert mechanisms was tested over a series of weeks, and in the context of a project, is described in Chapter Six.

Chapter Six. Second Evaluation

6.1 Study Objectives

The goal of the first evaluation was to gauge the reaction times and impressions of participants as they worked on projects unrelated to the alerts. However, a more thorough study was needed – one in which the alerts meant a broken build. The goal of this longer and more structured study was to gauge the effect on the developers.

Unlike the first evaluation, the developers were all working on a single project, AgilePlanner. Instead of the alerts being random and mixed, one single alert mechanism was used for each part of the study. The alerts were not random either – an alert occurred only if the Ant script compiling, deploying and testing the software resulted in a broken build. The purpose of this study was thus to assess the effect these notifications had on the development team as they worked on developing a piece of software together. Moreover, the goal was to evaluate the team's overall impression of BuildBot as a tool in the context of agile software development.

6.2 Study Description

As in the first evaluation, three types of notifications were tested. For Part I, an email was sent to only the developer responsible for the last code commit and the

research facilitator after every commit. The project manager was sent an email only if the change resulted in a build state change (from unsuccessful to successful, or vice-versa).

For Part II, the *Java Lava Lamps* were installed in the laboratory and the lamps were powered on or off depending on the build result (as described in Chapter Two: Previous Work). This was accompanied by an email, identical to Part I.

For Part III, the lava lamps were taken down and BuildBot was placed on its power station. Email was also sent in an identical fashion to Parts I and II.

6.2.1 Participants

The formal evaluation involved an agile team in an academic environment developing an application called AgilePlanner. This software has been under development for about 48 months, by both interns and graduate students.

There were two groups involved in this study: the *developers* and the *observers*. The eight developers were those actually working on the AgilePlanner. There are three full-time developers and five graduate students working part-time on the project. Five of the eight developers had used an email-based system of build notification for previous projects; two had no experience with build notifications. All developers had used Java for a year or more. Five of the seven developers had used JUnit for a year or more; one had used JUnit for two months, and the other had not used it at all. Of these developers, all

but one participant used some kind of email notification program (e.g. Google Talk, Adium, MSN Messenger, etc). Email thus served as a alert mechanism for these developers, and as a polling interface for the one developer not using any such program.

The observers were those individuals co-located in the laboratory but not actually participating in development. These individuals were able to observe the effects of the different notification mechanisms on the developers. There were six observers, but not all were available for the entire length of the study, so only those who were in the laboratory for the full length of each part were included in the results for that part. Because the observers were not directly involved in the project, their knowledge of the build was based solely on cues within the lab, including conversations between developers, or build notifications. They were included to get an outsider's view of the project and its build state – possibly a co-located non-developer who may be interested in the build status, or a customer.

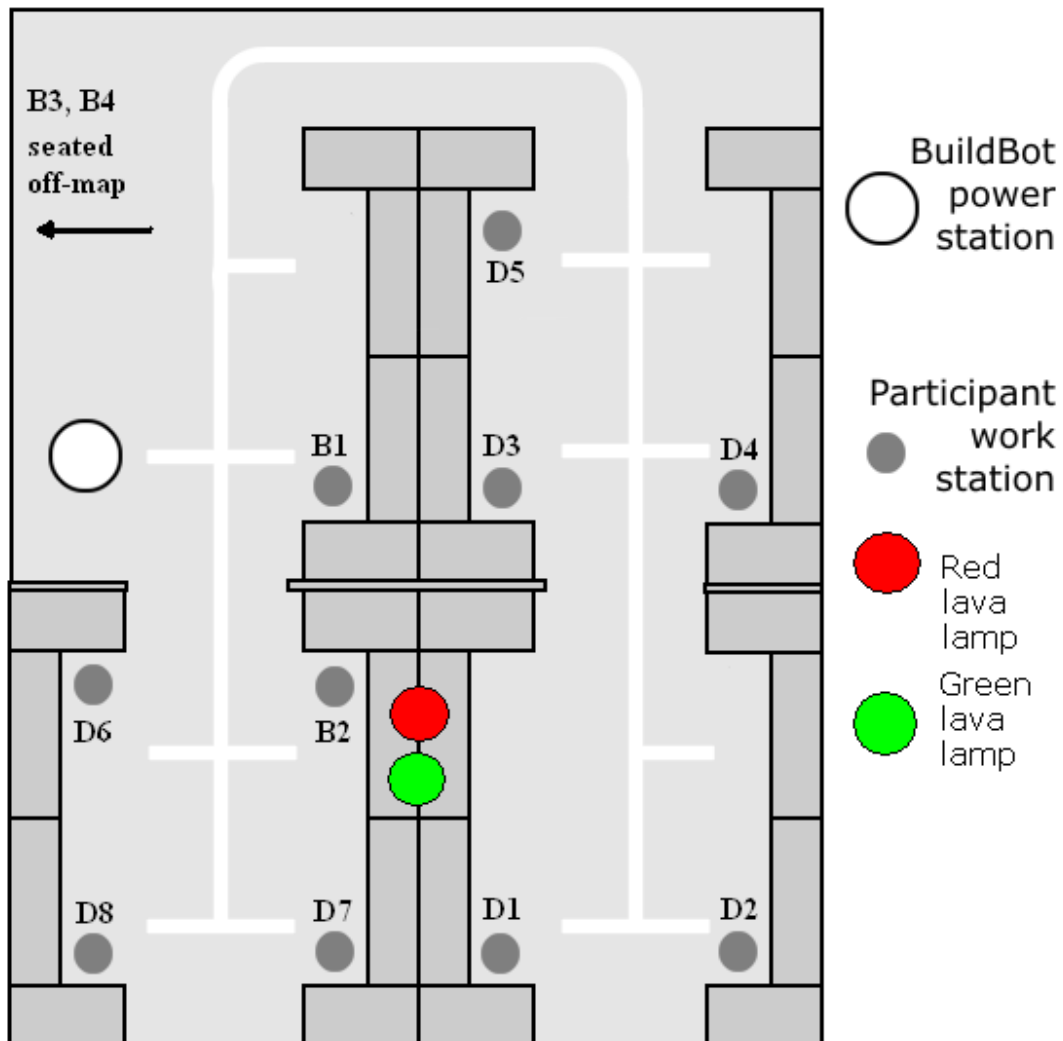
6.2.2 Scheduling

The study took place on weekdays between the hours of 09:00 and 17:00.

The study was partitioned into three parts. Part I of the study tested the email notification alone. Part II of the study involved the *Java Lava Lamps* combined with the email mechanism in Part I. Part III of the study tested BuildBot as well as the email notification in Part I.

6.2.3 Location

The study was performed in the same area as the first evaluation, but the robot's power station was moved to a different location, as illustrated in Figure 6.1. The station was moved to balance the time the robot took going to each desk. With the setup in the first evaluation, in which the station was situated in a nook between Developer 1 and Developer 2's desks, going to Developer 1's desk only took a minute, where moving all the way over to Developer 8's took substantially more time – 10 or more minutes. The top of the map is a hallway so there was a danger that people would trip over the robot if its station was placed there, so it was moved to an open area near Observer 1's desk.



**Fig.6.1: The layout of the laboratory. D indicates a developer and B indicates an observer (O is not used because it can be confused with the numeral 0)
The lava lamps and robot were tested independently of each other and were removed during the days they were not being used.**

Some individuals had a better line of sight to the lava lamps than others. Example lines of sight from different developers are shown in Figure 6.2.

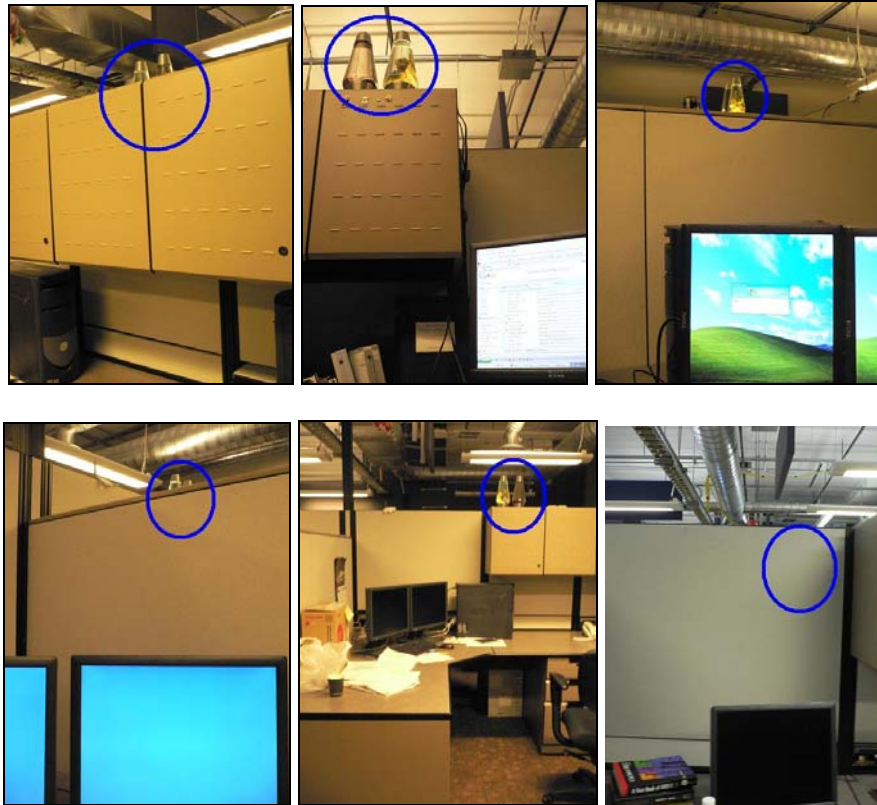


Fig.6.2: Line of sight from different developer workstations. From some workstations it is possible to see only the caps of the lava lamps, from some they are blocked entirely, and from others they are visible only by turning around.

6.2.4 Implementation

A continuous integration server was already in place, using an Ant script to compile, deploy and test committed code. The developers, as well as the project manager, were being alerted only to build breakages via email (sent via the Anthill continuous integration suite [Anthill 2007]). The application was deployed on a JBoss web server (JBoss 2007).

6.2.4.1 Guidelines for Developer Participants

The development team who participated in this study had been using Agile methods to develop the project. Some guidelines had to be enforced for the sake of the study.

As with the first evaluation, the participants were allowed to work as they would normally in order to have the least impact on the development effort. This includes sometimes pair-programming, using an email notification program, wearing headphones and/or listening to music. This altered the chances of some participants noticing the different notification mechanisms, but was done in order to observe the developers working in their environment. Some developers preferred to work at home, but were requested to work in the laboratory whenever possible.

Occasionally, when developers working on a project do not check the build status before they check-out code or commit new code, they can inadvertently cause problems. Specifically, BuildBot will go to the desk of the last person to break the build. If Developer A commits bad code in Class X that breaks the build, and Developer B uploads working code for an unrelated class Class Y shortly after, BuildBot will go to Developer B's desk, because that developer was the last person to submit code that resulted in a broken build.

To prevent this, a *commit token* was introduced. The commit token used in this study was a stuffed red mouse (Figure 6.2). It was hoped that this kind of playful commit token would also add to the atmosphere of fun and playfulness in the development environment. However, during the second evaluation, the number of developers was small enough so the commit token was not actually used. There were no conflicts between commits during the evaluation.



Fig.6.3: A developer working with the commit token (left), and the commit token itself (right).

For the commit token to be effective, the following guidelines were set:

- In order to commit code, a developer must have the commit token. They must acquire it from the developer who has it in his possession.
- That developer with the token may not release the token to another developer unless the build is stable and passes all tests.

6.2.4.2 *Java Lava Lamps* Implementation

As in the first evaluation, the lava lamps were displayed in a prominent position in the development area so that any developer could see it from their desk, although some had to turn around to see them, as they were facing the opposite direction. In the first evaluation, the lava lamps were controlled manually via remote control, but in the second evaluation they were automated to show the build status.

To integrate Java Lava Lamps into the development process (shown in Figure 6.3), the continuous integration server has a wireless interface module plugged into its serial port. This module sends a signal (to supply or deny power to one of the lamps) to the x10 transceiver. The transceiver is connected to one of two lava lamps, and relays the signal to the x10 module connected to the other lamp. Two signals are sent every time continuous integration occurs – the first signal turns one of the lamps off and the second turns the other lamp on.

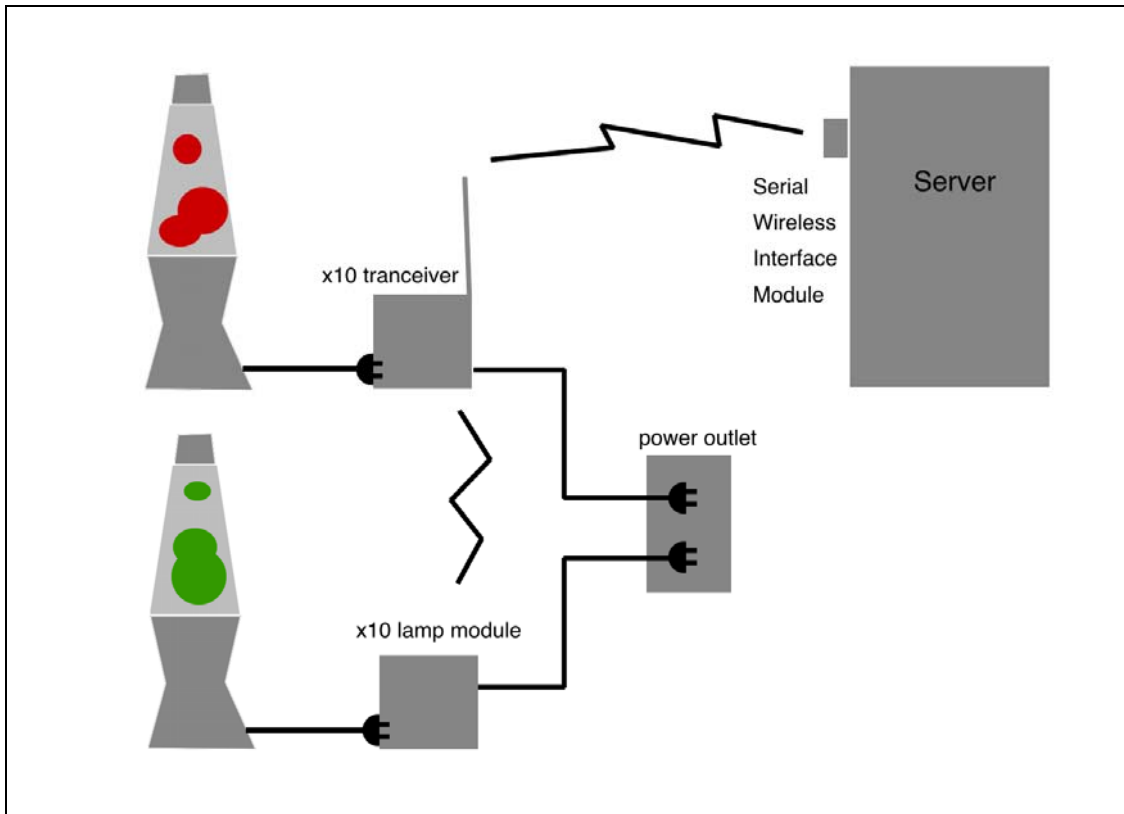


Figure 6.4. Setup for *Java Lava Lamps*.

The end result is that only one of the lamps has power at any given time. In the case of a successful build, the green lamp is turned on and the red is turned off. In the event of a build break, the red lamp is turned on and the green is turned off.

An x10 controller (x10, 2007) was used to control power to the lamps, and the software was written in Java, using Michel Dalal's Java library (Dalal, 2003) (this library takes care of the commands in the form of bytes being sent to the tranceiver).

6.3 Evaluation Results

The following section outlines results from each part of the study in detail, and then the results of the entire study. The discussion of the results will follow in Chapter Seven.

In the first week of the evaluation, a total of ten commits were recorded. In the second week three commits, one of which was a failure, were recorded. The third week saw ten more commits, one of which was a failure.

6.3.1 Results for Part I: Email Only

Eight developers and four observers participated in Part I of this study. The results here are from the Part I post-questionnaires and the interviews for each participant.

There were three main ways a participant could become aware of the build status. They could receive an email (if they committed code), check the build status page, or hear from other people about the status of the build. Five of the eight developer participants indicated they were notified more effectively by email than by polling the build status page. Two observers overheard conversation about the build status. Two of the developers did not receive emails, did not check the build status page and were not

alerted by other people about the state of the build. They were totally unaware of the build status.

Six of the developers indicated that they found the email fairly distracting or very distracting. Two of the respondents indicated they did not find the email distracting or found it a bit distracting.

Three of the eight developers and one of the four observers indicated that their ability to perceive things going around them was affected – they listened to music with earphones while working. Generally, those who were listening to music were not as aware of the build state as those who did not listen to music.

The data for the first week is shown in Figure 6.4 below.

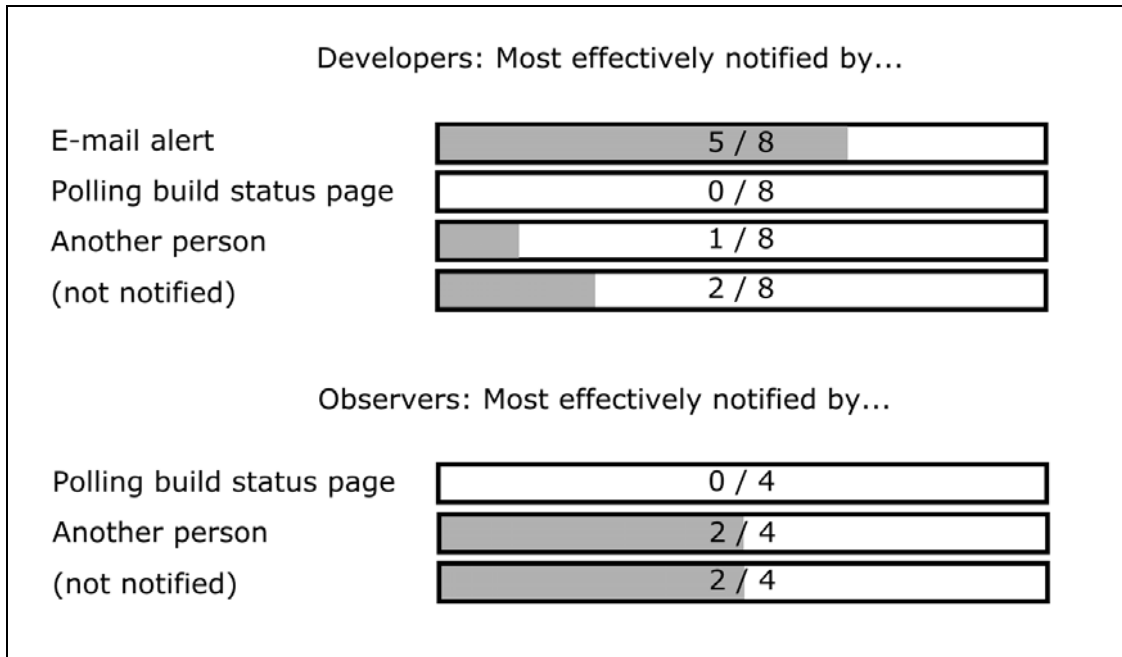


Figure 6.5 Part I Results - Notification

6.3.2 Results for Part II: Email and *Java Lava Lamps*

One of the part-time developers was unavailable, so seven developers and five observers were included in the second part of the study.

For the developers, half were better notified by email and half by either the colour change or movement of the lava lamps. While some participants did check the build status page and some were alerted by other people, all participants were alerted more effectively by either the lava lamps or by email. These results are shown in Figure 6.5 and 6.6 below.

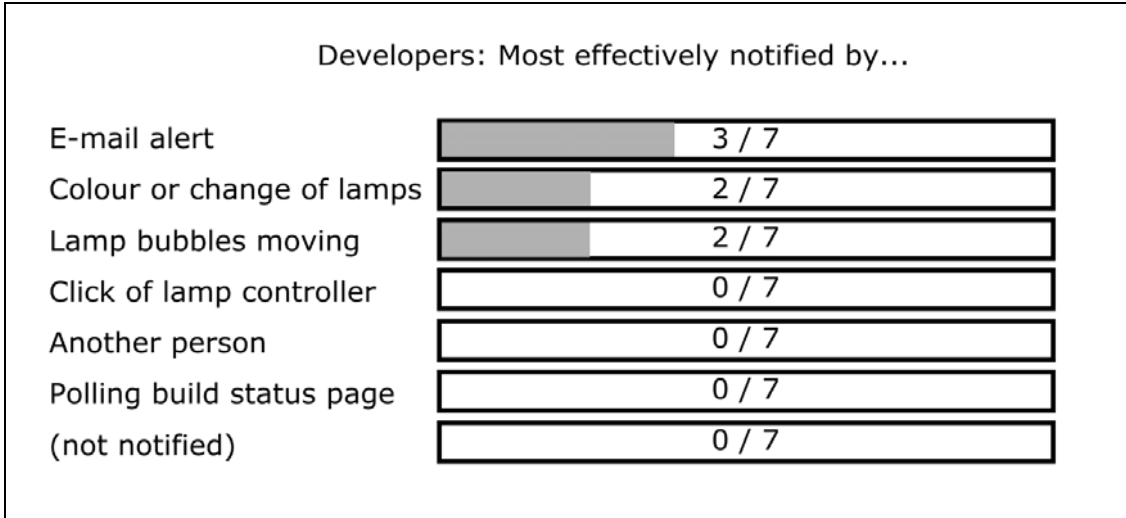


Fig.6.6: Part II Results - Notification – Developers

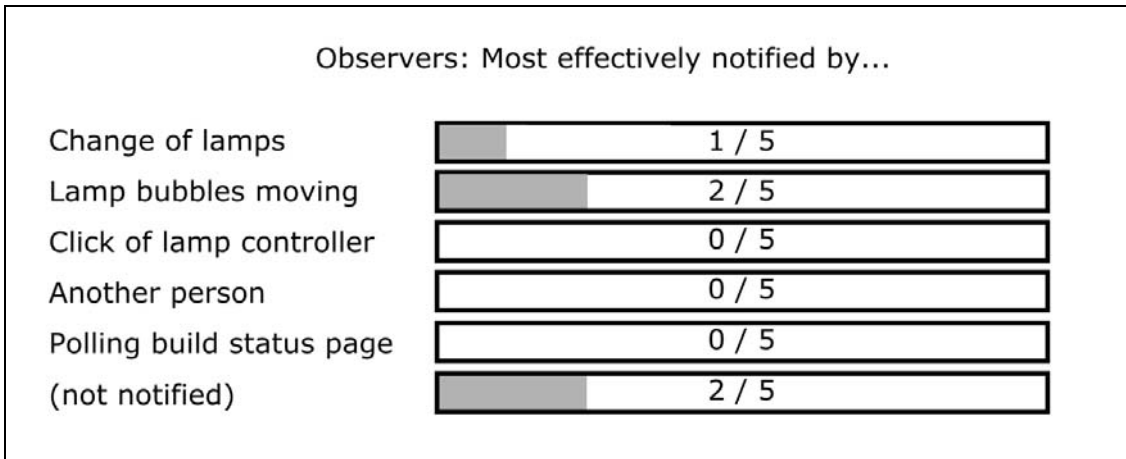


Fig.6.7: Part II Results - Notification - Observers

The developers generally found neither the sight nor the sound of the lava lamps distracting. The same was true for the observers, as they were generally too far away to be distracted by them.

Three out of the seven developers indicated they had a clear view of the lava lamps from their workstation, and three indicated a blocked view (Figure 6.7). Developer 7 and Developer 4 indicated they had blocked view of the lava lamps but received emails on alert via an email notification program (such as Google Talk, MSN Messenger, etc). They were thus able to keep track of the build that way. Developer 11 also polled the build status page for updates.

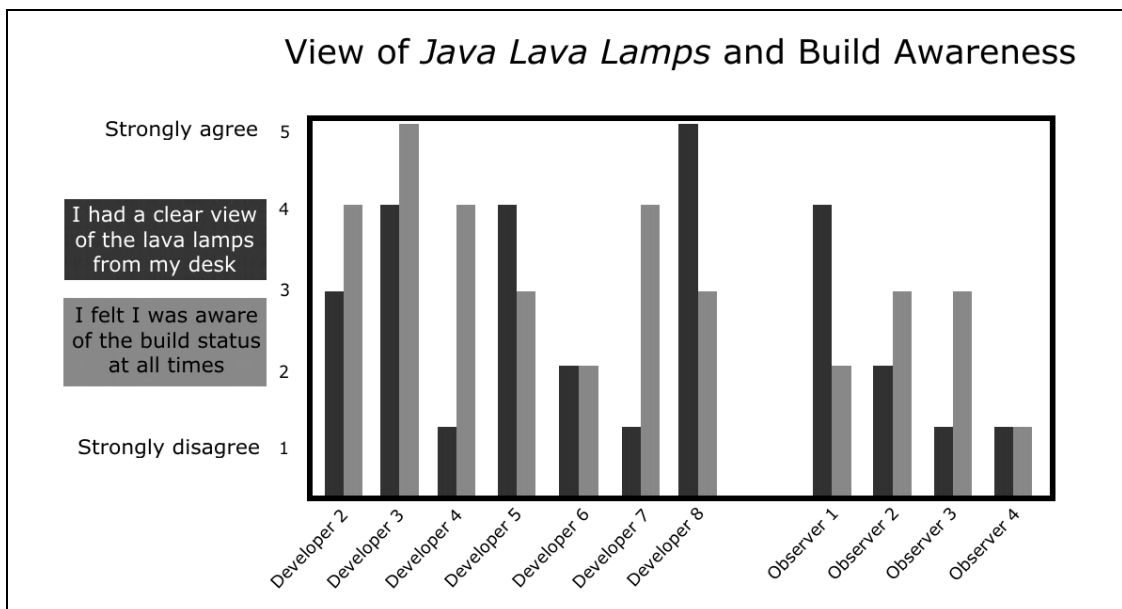


Figure 6.8: Part II Results - View of the Java Lava Lamps from workstation and awareness of the build.

As shown in Figure 6.7, there is a loose relationship between a clear view of the lava lamps and the awareness of the build. In general, if the participant had a blocked view of the lava lamps and was listening to music, he reported a lower awareness of the build. Conversely, if the participant had a clear view of the lamps and did not listen to music while he worked (Figure 6.8), a higher awareness of the build was reported.

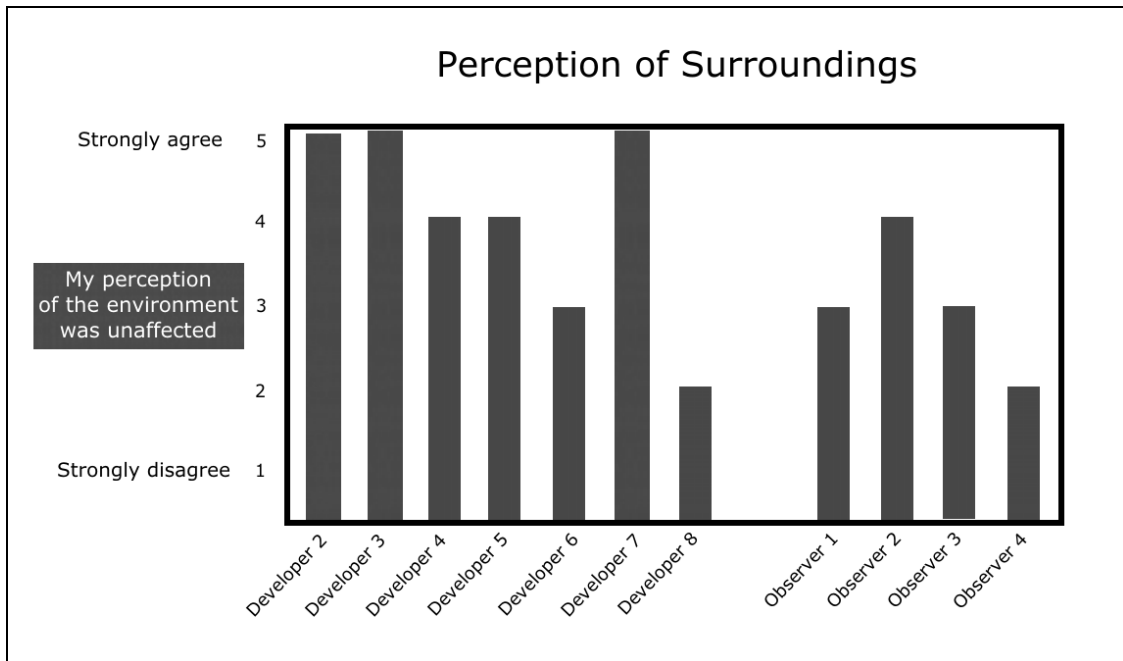


Fig.6.9: Part II Results – Perception of Surroundings

It is essential to examine each of the participants’ experience over the entire study. For example, participants who committed more often (the full-time developers 3, 4 and 5) had a better idea of the build since they committed code more often thus were the recipient of an email notifying them of the build status. Observers, not being part of the development process, naturally have a lower awareness of the build.

Some developers and observers indicated that they were able to see the lava lamps more easily when entering or exiting the room since some individuals were unable to see the lava lamps from their desk due to the cubicle layout of the laboratory. Nine of the participants indicated they found that the lava lamps contributed to a fun atmosphere

(Figure 6.9). Even though some participants could not see the lamps, they still felt that their presence made the development environment a more fun place to work.

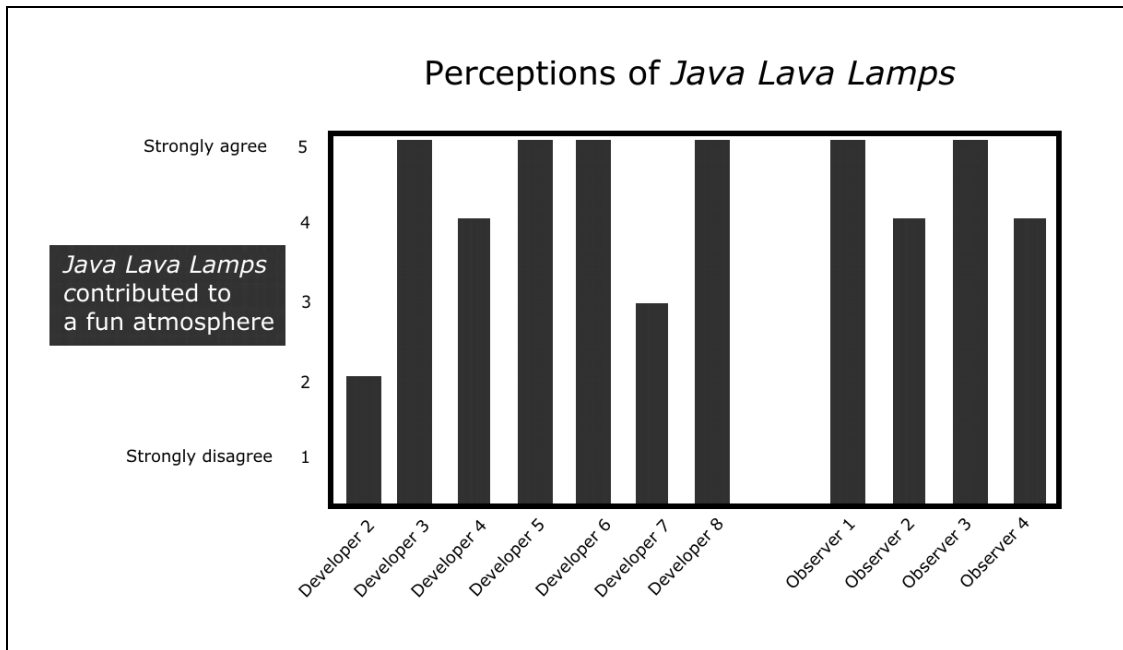


Fig.6.10: Part II Results - Perceptions of Java Lava Lamps

Observers, situated quite far away from the lamps and out of visual range, did not find the sight nor the sound of the lava lamps distracting. Developers, especially those situated near the lava lamps indicated they were mildly distracting; these results are shown in Figure 6.10.

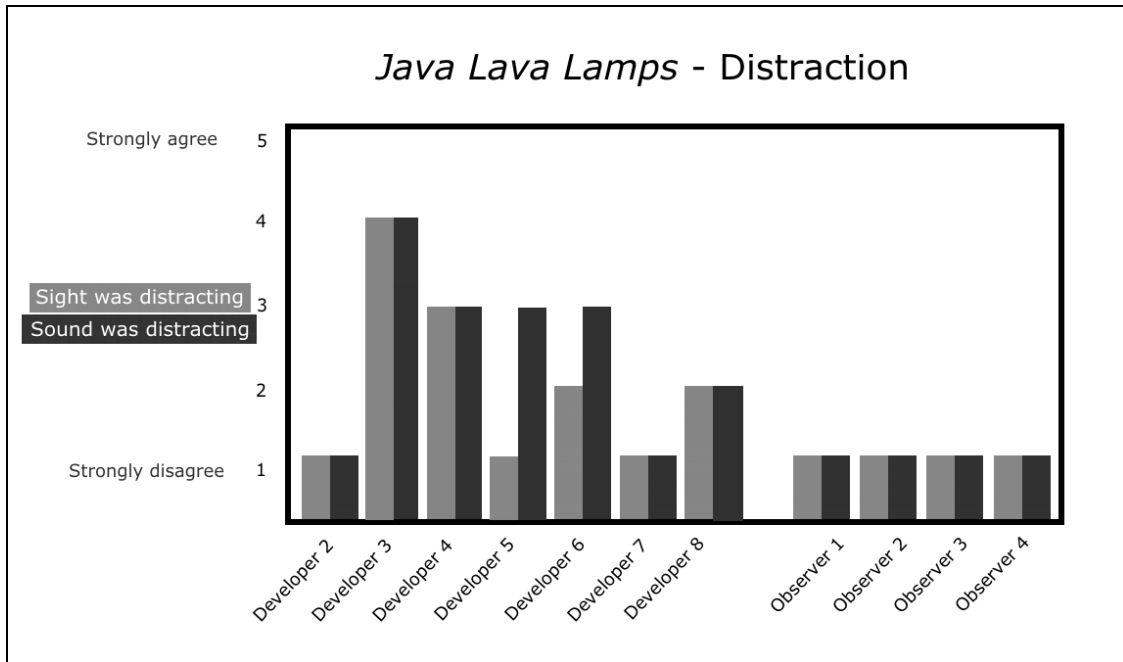


Fig.6.11: Part II Results - *Java Lava Lamps*' audio and visual distraction

Overall, the lava lamps contributed to a fun atmosphere generally without distracting the participants – though this may have been due to the novelty effect. Due to the cubicle layout of the development laboratory (and thus some of the participants being unable to see it), this limited its effectiveness in this environment.

6.3.3 Results for Part III: Email and BuildBot

During the third part of the study, the lava lamps were removed from the development area and BuildBot was placed on its power station. There were eight developers and four observers who participated. Developer 1 was in the lab at off-hours

and so did not have a chance to observe the robot, but was able to make observations about it from the first evaluation.

Only two participants of the twelve felt they were aware of the build at all times, as shown by Figure 6.11. This may have been due to the lack of constant, ambient communication about a positive build status – besides the build page, only by getting up to find the robot could they find out the status on the build.

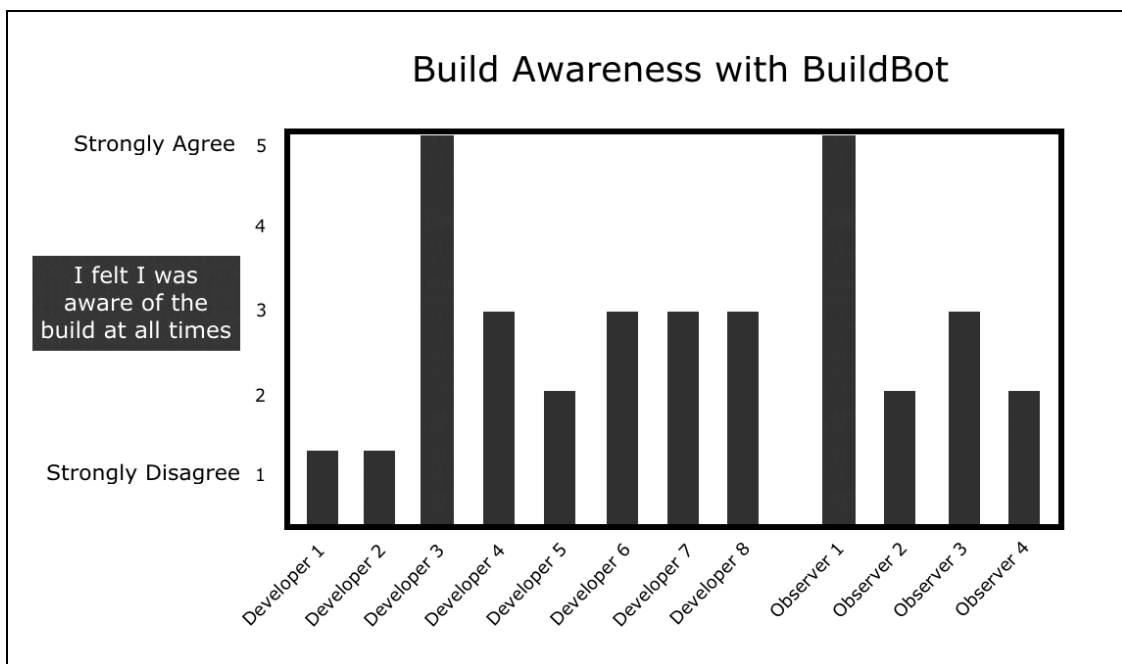


Fig.6.12: Part III Results – Build Awareness

Developers were again asked about their awareness of their surroundings, and the responses are shown in Figure 6.12. Unlike the lava lamps, which did not use sound to convey build state, BuildBot used sound as well as motion.

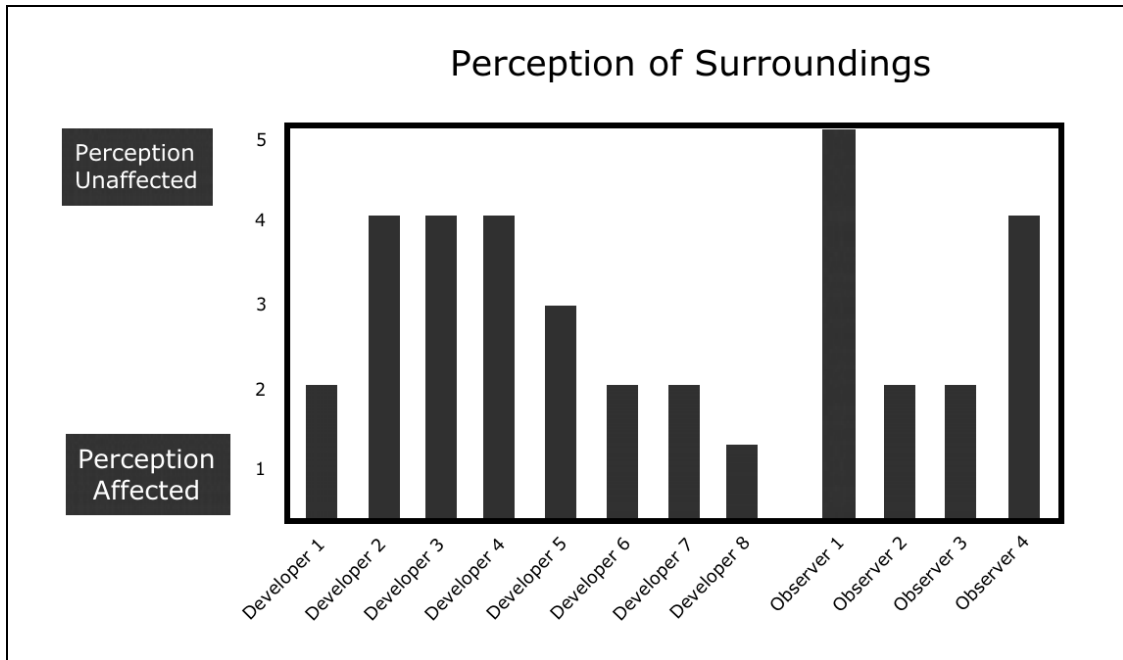


Fig.6.13: Part III Results – Perception of Surroundings

From the interviews and the questionnaire, it seemed that generally, those who found the robot to be a distraction (from sight, sound or both – Figure 6.13) did not find it contributed to a fun atmosphere (Figure 6.14). There were those who enjoyed BuildBot despite being distracted, but these participants may have been more willing to accept the distraction, or be able to work despite such distractions.

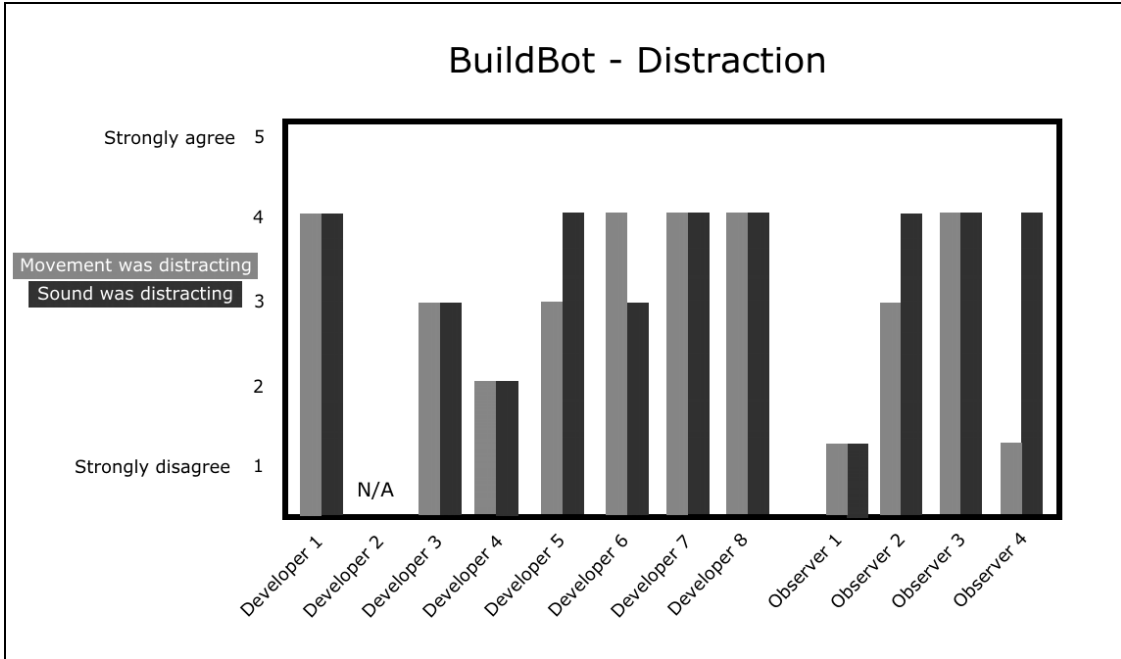


Fig.6.14: Part III Results – BuildBot – audio and visual distraction

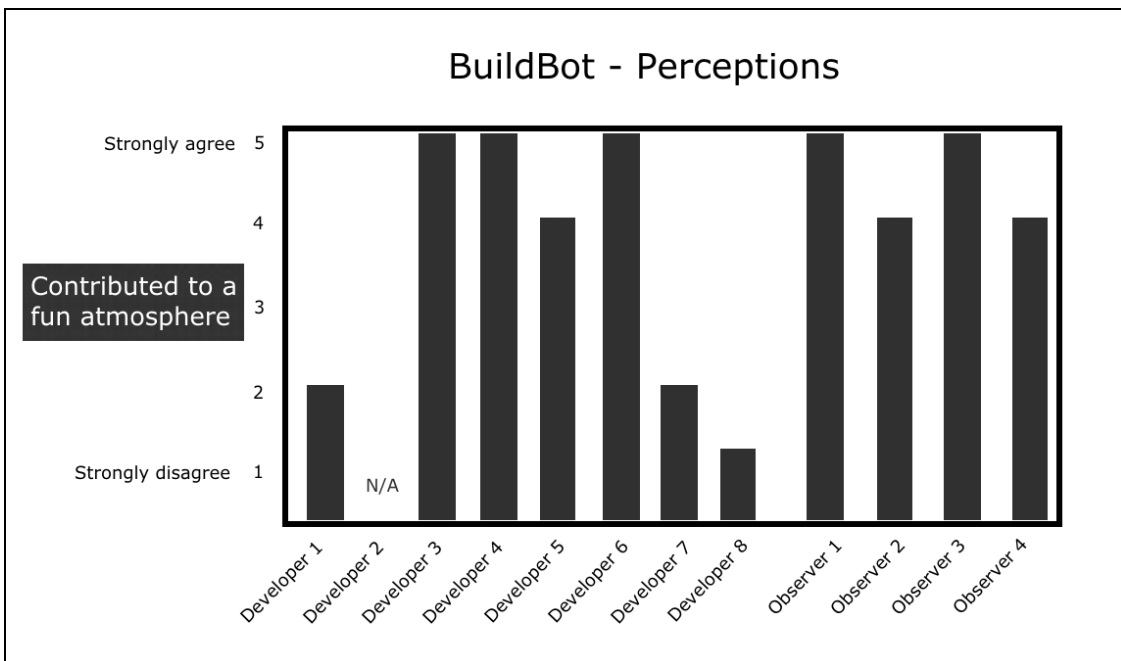


Fig.6.15: Part III Results –Perceptions of BuildBot

6.3.4 Final Results: Comparing All Three

After the third week, participants had seen each of the notification mechanisms in action and were able to compare them. Each participant indicated what aspect of the notification devices notified them most effectively. Email notifications were indicated by 4 of the developers (one developer was notified by another person who received the email), build status page notifications were indicated by one person, and two people indicated the robot most effectively notified them (Figure 6.15).

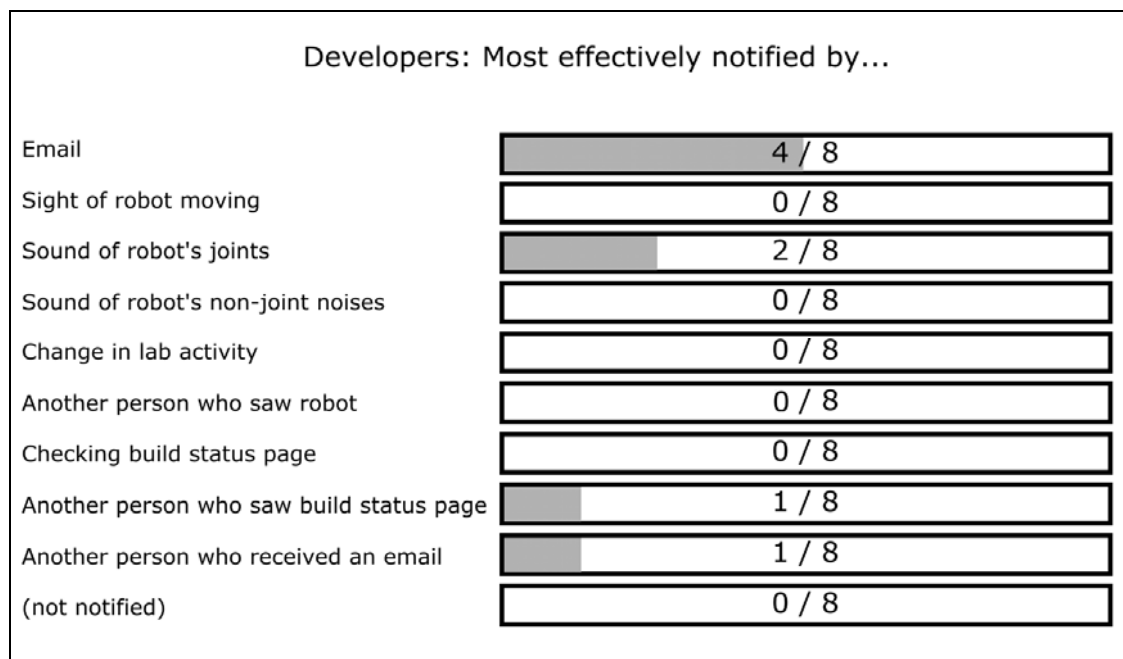


Fig.6.16: Part III Results - Developers

The observers were all notified by sound, either from the robot’s joints or from the non-joint noises (such as the initial sound notification, or the growling noise the robot makes when it arrives). Figure 6.16 illustrates this.

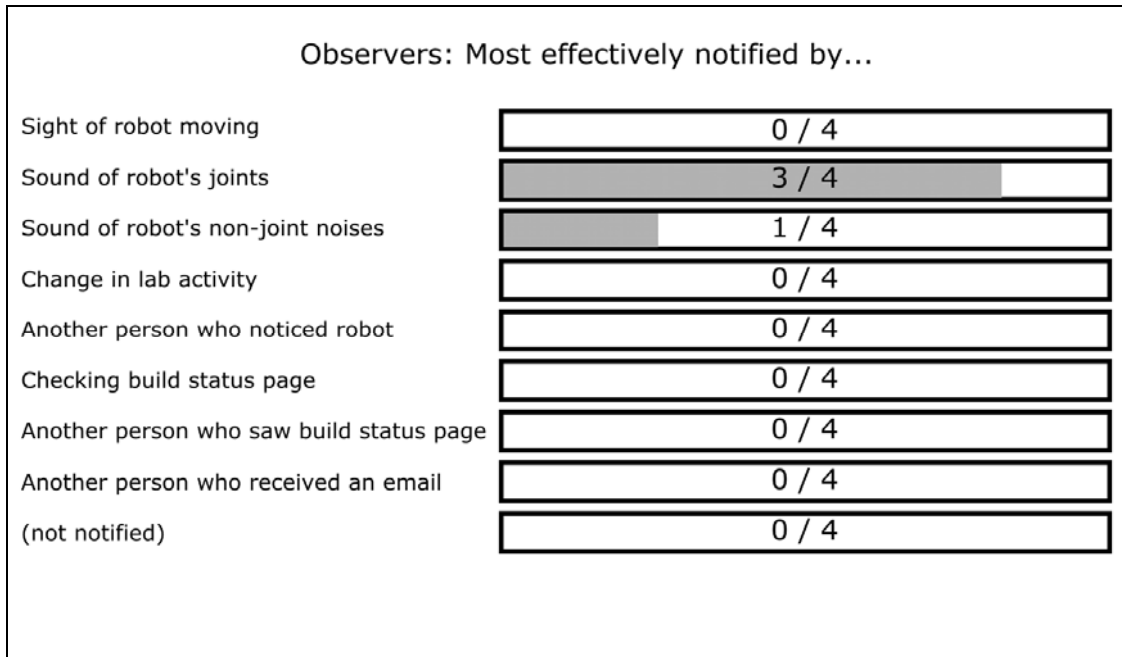


Fig.6.17: Part III Results - Observers

Some participants indicated that the robot added to a fun atmosphere, even though it was a bit distracting for them. They welcomed the diffusion of a potentially tense situation. Others did not find the robot very fun at all, and found the distraction very unwelcome. Some liked both email and BuildBot equally because they offered different but important advantages. Generally, the participants who preferred email were those who worked at home as well as in the lab. Figure 6.17 illustrates the developers’ preferences.

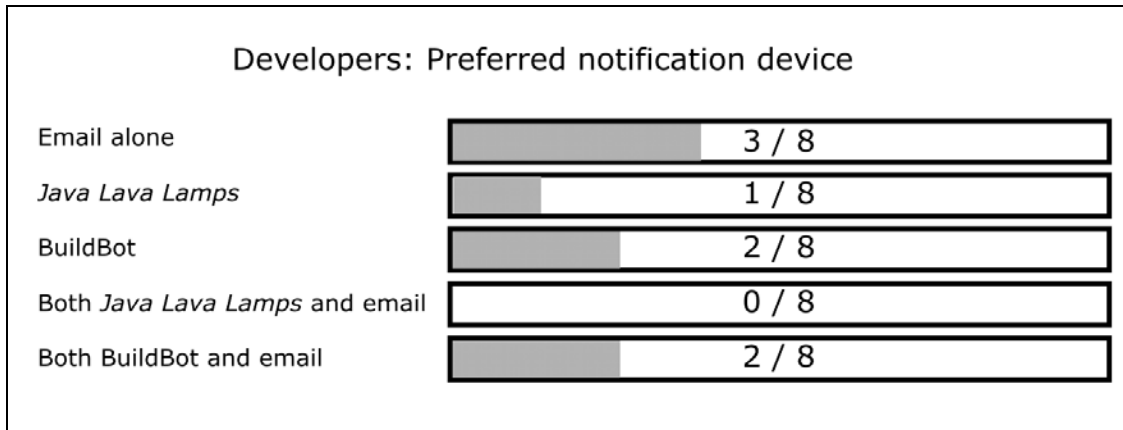


Fig.6.18: Part III Results – Preferred notification device - Developers

Figure 6.18 illustrates that the observers had a more positive view of the robot. This may have been due to their distance from the robot's path – they were less affected by the joint noises. Also, the possibility exists that because many of them had not seen the robot in action before, the novelty effect would give them a more positive impression. Also, since none of the observers received any emails, none of them chose that option.

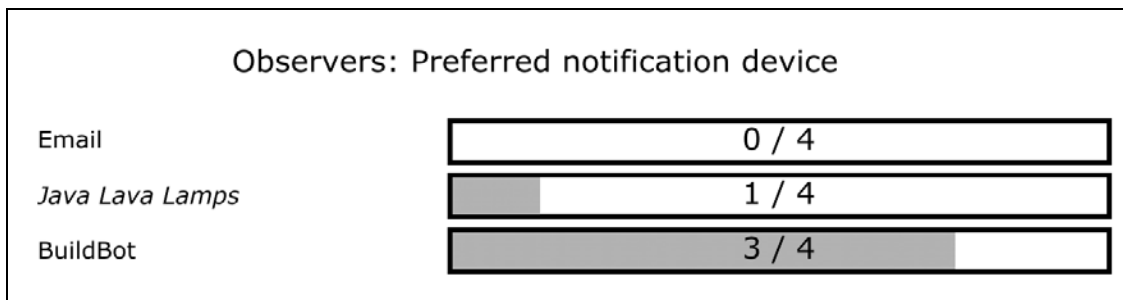


Fig.6.19: Part III Results – Preferred notification device - Observers

The qualitative results below were taken from interviews with the developers and observers who participated in this study. The quantitative results were obtained through participants' responses to questions in Likert scale format.

Throughout the entire study (all three parts), participants could see the build status anytime by checking the build status page. This is a pure polling mechanism, and not one participant used it. According to Developer 1, *“going to the website is not my idea of a notification system. It defeats the purpose.”* During the first week, even when the developers did not commit and thus were not receiving emails, they did not check the build status page.

There were also concerns about the reliability of the Ant scripts used. Developer 2 was leery about trusting notification systems: *“in my experience with Ant scripts, they're so flaky that I have a hard time trusting that everything went to fruition 100% of the time. Just because the robot's not moving, doesn't mean the build status is 100% correct.”* Developer 2 also added that *“I don't trust Ant scripts. A lot of the time they can die silently.”*

6.3.4.1 Positive and Negative Aspects of Email

Email was popular with the developers because of the information that each email contained. As shown in Figure 2.4 (in Chapter Two), the emails are very detailed, showing thrown exception messages, failed tests and other useful information. Developer

3 liked that “[email] elaborates in detail, like what went wrong, where”, and similarly, Developer 4 thought that “email has more information in it” to help them solve the build breakage.

Another advantage of using email is that it is, barring server or connection problems, instantaneous and direct. Email recipients can receive the message instantaneously, especially if they are using an email notification program (like MSN Messenger or Google Talk). This takes advantage of email’s dichotomy as a polling mechanism that can also act as an alert. Developer 5 expressed that “as soon as we get an email, we get a popup message, and we like immediately get to know that there is a build fail, because it’s written in that popup.” Knowing this information instantaneously is important to the other developers, as Developer 2 articulates: “I want to know right away if I committed some code that broke some tests, or something bad.”

Participants also liked that email was not dependent on location; according to Developer 1, “The email was the better of the three because I don’t have to be in the lab to get notified if something goes wrong.” Developer 2 agrees: “the obvious benefit is that I can be at home and receive an email, I can be in China and receive an email, I can be at my desk and receive an email. No matter what I do I can always receive emails to make sure I didn’t break it.”

Additionally, emails are silent and non-interfering with others in the same workspace. Developer 1 said “What I liked about the email notification is that it’s non-

intrusive. It shows up in your inbox if you've got an email notification system." For other developers working in the same area on different projects, they will not be distracted by information transmitted via sight or sound.

Email also has some disadvantages to it, such as its dependence on the user running email programs and notification clients. Developer 6 expressed that *"I might not perceive that there is new email because I don't check my email box from time to time. Every 5 minutes? No, I don't do that, maybe every hour. So if the build is failed I didn't know that."* Potentially, if a developer is not running an email alert program, they may be unaware of a critical build failure.

A danger with email is the potential for it to become spam. Developer 4 liked emails but expressed that *"too many emails make me crazy"*. Even though a positive email message to everyone informing them of a passed build may be appreciated, Developer 8 would caution the overuse of email in this way: *"I think it would be annoying, to be more than honest. It would leave me with a sense of being spammed."*

6.3.4.2 Positive and Negative Aspects of Java Lava Lamps

The lava lamps are a notification mechanism that diffuse information within a wide area without being intrusive. Developer 2 noted that *"It's a really cheap thing, you can just set it up, download your open source software, you set it up, and away you go. Any company can do that."*

The lava lamps were also a fun addition to the lab without being distracting. The participants generally liked their appearance, and thought they added fun to the development environment. They were also quiet and unobtrusive.

Unfortunately, many developers and observers did not immediately notice the lava lamps. The office is set up in an ‘open cubicle’ fashion, with four people sharing one large cubicled area. The walls are six feet tall and opaque. At each workstation, there are cabinets that can potentially block the view of the lava lamps. Also, some developers had their backs to the lamps. This makes the lamps less ambient than they should be.

Developer 1 said that *“I have my back to them, and they’re above my eye level when I’m sitting, so I have to physically turn around and look up”*, and Observer 2 noted that *“they needed more flashy things to catch your attention.”*

Similarly, since the lava lamps were only in one small area in the room, they were unfortunately not very noticeable. The bubbling was not noticed by any of the developers at all – as Developer 3 expressed, *“I’m working on my system and I don’t get attention to notify that lava lamps are red or green. That’s why the lava lamps didn’t help me much.”*

Developer 8 said that it was no problem to just turn and look at the lamps: *“it was really accessible because I could see the lava lamps and in a second I’d know, if I took the time to turn around and look behind me.”* For those out of sight range, the lava lamps became a device in need of polling because of their inaccessible information. Developer 5 noted that *“as long as it’s in sight range, it’s good enough. Otherwise, of course, no-one will*

get up from his seat and see the lamps every 10 minutes.” Some participants, like Observer 1, noticed the lamps when walking back to their workstation: *“if I’m away from my desk, walking back from desk, you know they’re very visible so I can really notice them.”*

Developer 4 liked the lamps but commented, *“funny but information here is limited. Just like the lava lamps only have two colours – green and red – that demonstrates 0 and 1, not as good as email. Email has many texts.”* In addition, Developer 6 expressed that *“[if] it is red, I’m not sure whether it is me [that was responsible for breaking the build].”*

Additionally, the lava lamps do not help a developer who is working from a remote location. Developer 2 noted that *“Lava lamps... won’t help me if I’m working at home. And a lot of people do work at home.”*

6.3.4.3 Positive and Negative Aspects of BuildBot

BuildBot was popular among some developers and observers because of the fun factor (and most likely a bit of the novelty factor as well). Developer 8 noted that *“I liked the robot because it typically made a bad situation fun. Breaking a build is never a fun thing, because you know you have to fix it, and it sort of added a playful edge to it.”*

Observer 1 said that *“[BuildBot] makes the atmosphere a little more fun, cos it’s kinda*

like a joke... *'ha ha ha, you broke the build'*” Developer 3 felt that despite the distraction, BuildBot was fun and *“a positive thing to relieve your stress.”*

Even though the robot was seen as distracting by some developers, not all found this distraction to be a negative thing. Developer 6 sees distraction to be a part of working in a lab with others: *“when I’m working, somebody may be talking, so that’s a distraction. I can put my headphones on.”*

The robot’s message was very accessible according to Developer 8, who noted that *“it’s accessible because it comes to you. In other words, it’s very visible when it comes to you, as opposed to the lava lamps, because you’re getting both sight and sound as a notification.”* Developer 2 expressed that *“BuildBot is cool because it’s a little bit different from what you see normally, and you can hear it coming, which is kind of nice.”* Observer 4 said that BuildBot *“[is] easy to notice, it’s interactive, it’s kind of novel”* However, when asked to elaborate about his impression of the robot should the novelty effect wear off, Developer 1 responded that *“initially it would be the novelty effect, and over time it would just become annoying, more than anything.”*, and after a few runs of the robot *“I would probably be cursing at the dog.”*

As Developer 1 notes, the obvious downside to BuildBot is the fact that it notifies everyone around the lab – whether the information is useful to them or not. Observer 4 noted that *“it does cause some commotion – if you’re focused on something, and trying to work on something, it can be distracting.”* Similarly, Observer 1 expressed that *“it is*

distracting. If it's in the far corner, that's like 5 minutes to walk there and 5 minutes to walk back, and you can hear the motors throughout the entire lab the whole time...

Observer 1 also suggested that an improvement to BuildBot might be to *"make the joint noises on the robot quieter"*.

A side effect of these noisy joints is that it causes commotion as people get up and *"[get] more restless... looking to see what it was"* (Developer 1). It thus distracts by joint noises and by people getting up to investigate the joint noises.

Developer 2 expressed concern with BuildBot's singling out of individual developers after a certain time period: *"what's the difference between a robotic dog giving you hell and a manager giving you hell? Like a manager may be a little bit angry but generally they're not."* Observer 3 was notified not just by the sound of the robot's joints, but *"also the commotion caused by other people in the same area who were laughing, pointing, giggling at said programmer [who broke the build]"*. While this is *"all in pretty good fun"* and *"a healthy embarrassment"* according to Observer 3, I believe this could cause some developers stress and could negatively impact the team.

An issue for some participants was the lack of positive notification. Observer 2 *"wasn't entirely sure at all times if there was a problem or not."* because there was no notification whatsoever if the build passed.

With the mixed reviews of BuildBot, companies may wonder about the feasibility of spending a potentially large amount of money on purchase and setup of it. Developer 2 articulates this point: *'I don't know how many 4-person companies are going to be able to afford an AIBO to have the BuildBot walk around. And out of the money they're going to spend – I think even a big company's going to say, "what's wrong with lava lamps?" The price of having a manager walk around and do it is probably cheaper than having BuildBot do it!'*

The original goal of BuildBot was to bring together the advantages of aspects of human-computer interaction and human-robot interaction to create a more effective build notification device for an agile software engineering environment. It was hoped that presence, sound, motion and fun could enrich the development environment and more effectively alert the developers to a broken build. The results of this second evaluation show that quantitatively, BuildBot is a more effective, but also a more distracting and a more complex, build notification mechanism. Qualitatively, some developers liked the idea of having a robot, but others found it an unwelcome addition to the environment.

Chapter Seven. Discussion

In this chapter, some questions with regards to the development environment will be addressed.

7.1 Development Environment Culture

7.1.1 What happened to the commit token?

The robot's behaviour is not equipped to handle a situation in which one or more developers commit new code to an already broken build. The robot's behaviour cannot determine which developer is responsible, and can only visit one desk at a time. The commit token was an attempt to avoid this situation during the second evaluation. A commit token is a common practice meant to restrict developers from freely committing code. In theory, it prevents multiple developers from compounding the bugs in the code, since commits can only be made to a clean build.

The group of participants had attempted to use the commit token before, but it had not been successful. Similarly, during this study, it was not used. During the study, it appeared that the commits were not done often enough to warrant the token, and thus the participants did not use it.

7.1.2 Should the development environment be fun?

Some participants preferred a quieter environment that is more conducive to productivity. According to Developer 1, *“you’re trying to work and suddenly you hear something moving, and it’s not your typical lab noise, so it was distracting”*. Developer 1 also stated that, in response to a question asked regarding the robot adding to a fun atmosphere, *“I don’t think work should be in a fun atmosphere. If I’m trying to be productive I don’t want to necessarily be having fun. When I’m distracted and not doing work, I’m annoyed. So it’s not really fun.”* On the other hand, according to Developer 3, *“I think your work environment should be kind of a fun environment, and then you will remain positive and you will work better.”* For Developer 5, not all distractions (and specifically, the ones due to BuildBot) are negative: *“It’s like the distraction means you go from a serious mood to a fun mood for a couple of minutes.”*

7.2 Notification Device Alternatives

Many developers liked the idea of the lava lamps but were unable to see them properly because of the cubicle walls. Observer 4 noted that a possible improvement to the lava lamps would be to *“make them visible from everywhere... make them bigger, brighter, or something.”* A possible candidate is a light source that can be visible from many or all parts of the laboratory. An inexpensive colour-changing rope light affixed to the ceiling may be another ambient device to try in a future study. Another candidate would be a smaller set of lights at each developer’s desk.

Another concern brought up by Developers 1 and 2 was that the broken build notification does not provide indication that the problem is being fixed. Furthermore, the timing could be changed to accommodate developers attempting to fix the broken build. Developer 1 stated that the lava lamps are not “*a very positive notification mechanism*” and elaborated: “*I’m really not a project manager, and I don’t know how much they care about the status of the build in the middle of a development cycle. If you’re always adding new code in the middle of a development cycle, if the lava lamp’s red but you’re working on a fix, all the project manager who just walked in the room knows is that the build’s broken, but not that you’re working on a fix for it... It might be that the project manager will come in and interrupt your work and your train of thought when you’re trying to solve the problem, and then you block that train of thought.*”

Chapter Eight. Conclusion and Future Work

Robotic technology is not at the point where BuildBot is feasible for use in an agile software development environment, and it is questionable whether it will become feasible in the near future. Judging by the feedback from the participants in this study, the robotic implementation as presented herein is too complex and costly to be a realistic option for development teams in industry. Moreover, there are other less obtrusive means of notification, such as ambient illuminated rope lighting or virtual system tray alert, that are valid alternatives and could be tested in future studies.

The developers seemed to like all three notification devices, but BuildBot had the most potential for strife due to the distraction factor. Given the cost of the robot, the AIBO's loud joint movement, and the inherent complexity of the system, it is at this point not feasible to introduce to an agile team. Perhaps, if BuildBot is improved and implemented on a more advanced robotic platform, it may become feasible for use in such an environment.

8.1 Conclusion

“...Because that’s the biggest part, right? It’s awareness. It’s not the fact that I need to fix it in this amount of time. It’s to be aware of the problem.”

- Developer 2

There is much work to be done in the area of continuous integration notification mechanisms and its effect on the agile software engineering environment. There are many different mechanisms that can be tested in further studies. Examples include pagers, visits from project managers, multicolour rope lighting, or system tray alerts.

Notification of the build status of a project is very important – not only for team solidarity but for each developer’s own benefit. Developer 4 expressed that *“build notification is very useful because it will make me aware of the process of the whole project and the status. It is important and I like that.”* Three different notification systems were tested in this study – a totally virtual email notification, an ambient lava lamp system, and a mobile robot.

It was hoped that BuildBot could keep the development team focused on the task at hand, that is, to efficiently develop software whose components work together seamlessly. While the robot has the potential to increase morale in this way and to keep the social environment more engaging and fun for the agile developers, it depended upon the developers’ attitude toward their work environments. Those who liked a more fun,

lighthearted environment liked BuildBot despite its tendency to distract, but those who wanted a quiet environment conducive to productivity disliked the robot. If all members of the team are in favour of implementing BuildBot, that may be a good opportunity to implement it. However, to impose something as potentially disruptive on an agile team would not be in its best interests.

8.2 Future Work

A longer-term study is necessary to examine the long-term effects of different build notification mechanisms. The study should also be performed in an agile industrial environment.

In this study, only two states were shown on each of the devices: a successful build and a failed build. There are, however, different types of failures (Deng et al, 2007), and for this study they were all grouped into one category – failure. Giving developers more information about the failure via an ambient device may make the devices more useful.

A future improvement would be to add notification that the build is being fixed. For example, a third lava lamp could be used to signal this status. Developer 2 suggested a solution: *“For me, it seems a better approach would be I break the build, you send me an email, and I have a button I can click on here saying ‘I’m notified of this problem and I’m trying to fix it’, because I can’t fix every problem in ten minutes. And then, as long as*

I actually notify them that I'm aware of the problem and in the process of trying to fix it, then the dog shouldn't come and harass me, just because I can't do it in ten minutes."

As for BuildBot itself, there were several points of improvement that were mentioned by participants in the study. One complaint is that BuildBot takes too long to reach the workstation, irritating everyone on its way over. Many developers who did like BuildBot didn't like the slow movement. Developer 6 pointed out that "*[If] the dog is walking towards me, and maybe barking at me, that means I've made some mistake. I like that idea. But the problem for the robot is that is walking slowly, so everybody's waiting for that. So it's a waste of time.*" Developer 7 noted that "*I like to have a quiet environment, so that I can focus on my programming. I don't mean that there was a big distraction, but somewhat like a distraction. It took a long way for the robot to walk towards me or the other people sitting around me.*", and similarly, "*it's distracting, and if it walked faster it would be better. Because it's a long way, and you will hear the sound a long time if you don't have headphones on.*"

References and Citations

- Ambient Devices' Ambient Orb*, <http://www.ambientdevices.com/cat/orb/orborder.html>
- Anthill Pro* build management system.
<http://www.anthillpro.com/html/products/anthillos/default.html>
- D. Astels, 2003. *Test-Driven Development: A Practical Guide*. Prentice Hall, PTR, Upper Saddle River, NJ
- D. Astels, G. Miller, M. Novak. *A Practical Guide to eXtreme Programming*. Prentice Hall PTR, Upper Saddle River, NJ, 2002.
- S. Baker. *Agile in Action: Information Workspace*. 2005.
<http://www.think-box.co.uk/blog/2005/08/informative-workspace.html>
- K. Beck. *Extreme Programming Explained*. Addison-Wesley, 2000.
- M. Bohlen and M. Mateas. Office Plant #1. LEONARDO, 31:5, November 1998, pp 345-349. <http://www.lcc.gatech.edu/~mateas/publications/Leonardo1998.pdf>.
<http://www.ehiprimarycare.com/news/item.cfm?ID=2102>
- C. L. Breazeal. *Designing Sociable Robots (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2002.
- Cactus* <http://jakarta.apache.org/cactus/>
- JJ Cadiz, G.D. Venolia, G. Jancke, and A. Gupta. Designing and Deploying and Information Awareness Interface. *Proceedings of the ACM Conference of Computer-Supported Cooperative Work (CSCW 2003)*. ACM Press, 314-323.
- J.J. Cadiz, G.D. Venolia, G. Jancke, A. Gupta. Sideshow: Providing peripheral awareness of important information. *Microsoft Research Technical Report MSR-TR-200181*, 2001.
- CruiseControl. <http://www.cruisecontrol.sourceforge.net> (Accessed February 2007)
- Mike Clark's Build Automation Blog. <http://www.pragmaticautomation.com/cgi-bin/pragauto.cgi>
- A. Cockburn, *Agile Software Development: The Cooperative Game*, Agile Software Development Series, 2001, pp 70-80.
- E. Cutrell, M. Czerwinski, and E. Horvitz. Notification, Disruption, and Memory: Effects of Messaging Interruptions on Memory and Performance. *Proceedings of the IFIP TC.13 Conference on Human Computer Interaction (Interact 2001)*.

A. Dahley, C. Wisneski, and H. Ishii. Water Lamp and Pinwheels: Ambient Projection of Digital Information into Architectural Space. *Summary of CHI '98*, (Los Angeles, California, USA, April 1998), pp 269-270

M. Dalal's code library. <http://www.micheldalal.com>

K. Dautenhahn, S. Woods, C. Kaouri, M. Walters, K. Koay, and I. Werry. What is a Robot Companion – Friend, Assistant or Butler? *In Proc. IEEE IROS (2005)*, 1488-1493.

C. Deng, P. Wilson, F. Maurer: FitClipse: An Eclipse Plug-in For Executable Acceptance Test Driven Development, Proceedings of the 8th International Conference on Agile Processes in Software Engineering and eXtreme Programming (XP 2007), Come, Italy 2007 (Springer).

C. DiSalvo., F. Gemperle, J. Forlizzi, and E. Montgomery. (2003). "The Hug: An Exploration of Robotic Form for Intimate Communication." Ro-Man 2003 Conference Proceedings, San Francisco, CA, October, 2003

E-Health Insider, "Flashing 'orb' to help medication adherence". *E-Health Insider*, 04 September 2006. (Accessed March 2007)

M. Fowler, "Continuous Integration". May 2006.
<http://www.martinfowler.com/articles/continuousIntegration.html> (Accessed March 2006)

B. Friedman, P. Kahn, and J. Hagman. Hardware Companions? – What Online AIBO Discussion Forums Reveal about the Human-Robotic Relationship. *Proc. Of CHI 2003*, ACM Press (2003), 273-280.

L. Halvorson, "Another Ambient Orb used for continuous integration status". 30 June 2004. <http://weblogs.asp.net/lorenh/archive/2004/06/30/170089.aspx> (Accessed February 2007)

J. Heiner, S. Hudson, and K. Tanaka. The Information Percolator: Ambient Information Display in a Decorative Object. *In Proc. of UIST 1999*, pp. 141-148.

H. Ishii, C. Wisnesky, S. Brave, A. Dahley, M. Gorbet, B. Ullmer, and P. Yarin. ambientROOM: Integrating Ambient Media with Architectural Space (video). *Summary of CHI '98*, (Los Angeles, California, USA, April 1998), pp 173-174

JBoss Application Server. <http://www.jboss.org/products/jbossas> (Accessed December 2006)

JUnit. <http://www.junit.org/> (Accessed January 2007)

Z. Khan. Attitudes towards intelligent service robots. NADA KTH, Stockholm 1998.

- C.D. Kidd, Sociable Robots: The role of presence and task in human-robot interaction. MSc thesis, Massachusetts Institute of Technology, June 2003.
- S. Kiesler, P. Heinds. "Introduction to this special issue on human-robot interaction" Human Computer Interaction, 19, 1-8 (2004).
- C. Larman, 2004. *Agile & Iterative Development: A Manager's Guide*. Agile Software Development Series. Addison-Wesley, USA. pp 292-295.
- Manifesto for Agile Software Development*, 2005. (Accessed June 2007)
<http://agilemanifesto.org>
- D. McFarlane. Coordinating the Interruption of People in Human-Computer Interaction. *Proceedings of the IFIP TC.13 Conference on Human-Computer Interaction (Interact 2001)*.
- M. Mori, "Bukimi No Tani (the uncanny valley)", *Energy*, 7, pp. 33–35, 1970.
- NUnit*. <http://www.nunit.org/> (Accessed May 2007)
- R. Picard. *Affective Computing*. MIT Press, 1997.
- Pragmatic Automation: Bubble, Bubble, Build's in Trouble. 26 August 2004.
<http://www.pragmaticprogrammer.com/pa/pa.html>
- L. Rising, N. S. Janoff, "The Scrum Software Development Process for Small Teams," *IEEE Software*, vol. 17, no. 4, pp. 26-32, Jul/Aug 2000.
- P.N. Robillard, P. Kruchten and P. d'Astous 2003. *Software Engineering Process with the UPEDU*, Addison Wesley, USA.
- D. Saff and M.D. Ernst. An Experimental Evaluation of Continuous Testing During Development, In *International Symposium on Software Testing and Analysis (ISSTA '04)*, pp76-85, Boston, USA, July 11-14, 2004.
- A. Savoia. Developer Testing: eXtreme Feedback Devices for Software Development. 01 April 2004. <http://www.developertesting.com/archives/month200404/20040401-eXtremeFeedbackForSoftwareDevelopment.html>
- A. Savoia. On Java Lava Lamps and other eXtreme Feedback Devices. 26 August, 2004.
<http://www.artima.com/weblogs/viewpost.jsp?thread=67492>
- M. Scheef, J. Pinto, K. Rahardja, S. Snibbe, and R. Tow. Experiences with Sparky, a Social Robot. In *Proc. Workshop Interactive Robot Entertainment (WIRE)*, Pittsburgh, PA. 2000.

- Slashdot. Pragmatic Project Automation. 31 August 2004.
<http://books.slashdot.org/article.pl?sid=04/08/31/1634258>
- J.F. Smart. *Continuous Integration with Continuum*. 30 May 2006.
<http://today.java.net/pub/a/today/2006/05/30/continuous-integration-with-continuum.html>
- Sony AIBO home page. <http://www.sony.net/Products/aibo/>. (Accessed April 2006)
- M. Swanson, "Automated Continuous Integration and the Ambient Orb". 29 June 2004.
<http://blogs.msdn.com/mswanson/articles/169058.aspx> (Accessed March 2007)
- S. Thrun. Spontaneous short term interaction with mobile robots in public places. *In Proc. IEEE Int. Conference on Robotics and Automation, Detroit, MI. pp10-15. May, 1999.*
- E. Tira-Thompson, N.S. Halelamien, J.J. Wales, and D.S. Touretzky
 Proceedings of the Sixth International Conference on Cognitive Modelling, July, 2004,
 pp. 390 - 391.
- J. Wainer, D. Feil-Seifer, D. Shell, and M. Mataric. The role of physical embodiment in human-robot interaction. In *IEEE Proceedings of the International Workshop on Robot and Human Interactive Communication*, University of Hertfordshire, Hatfield, United Kingdom, September 2006.
- M. Weiser, J. S. Brown. "Designing Calm Technology", *PowerGridJournal*, v 1.01,
<http://powergrid.electriciti.com/> 1.01, July 1996.
- C. Wisneski, H. Ishii, A. Dahley, M. Gorbet, S. Brave, B. Ullmer, P. Yarin. Ambient Displays: Turning Architectural Space into an Interface between People and Digital Information. *Proceedings of the International Workshop on Cooperative Buildings (CoBuild '98)*, Darmstadt, Germany, February 1998, pp 22-32.
- x10 Firecracker power control kit. http://www.x10.com/automation/ck18a_s_ps32.html
 (Accessed May 2006)
- M. Xin., E. Sharlin Exploring Human-Robot Interaction Through Telepresence Board Games. In Proc. ICAT 2006, Lecture Notes in Computer Science, Volume 4282/2006, Springer (2006).
<http://www.cs.cmu.edu/~tekkotsu/media/tira-thompson-iccm04.pdf>
- J. Yamato, R. Brooks, K. Shinozawa, and F. Naya. Human-Robot Dynamic Social Interaction. *NTT Technical Review*, Vol. 1, No. 6, September 2003.
- J. Young, G. McEwan, S. Greenberg, and E. Sharlin.
 Aibo Surrogate - A Group-Robot Interface. *Demo, Adjunct Proc ACM CSCW 2006.*

Appendix A: First Evaluation Post-Questionnaire

Please circle the answer to the questions below (You can circle more than one if it applies)

1. What alerted you to the change in state of the build?

Lava lamps Robot

Other _____

2. How did you notice the change?

Sight Sound

Other _____

Please answer the following questions with a scale of 1 to 5, where 1 = *strongly disagree* and 5 = *strongly agree*.

The sight of the lava lamps was a distraction.

1 2 3 4 5

The clicking sound of the lava lamps was a distraction.

1 2 3 4 5

The sound of the robot was a distraction.

1 2 3 4 5

The movement of the robot was a distraction.

1 2 3 4 5

The email was a distraction.

1 2 3 4 5

The robot contributed to a fun atmosphere.

1 2 3 4 5

I can see the lava lamps from my desk.

1 2 3 4 5

My ability to perceive things going on around me was affected (eg. headphones, talking with other people, etc)

1 2 3 4 5

Appendix B: Second Evaluation Pre-Questionnaire

1. How long have you been working to develop AgilePlanner?

2. How much experience do you have with the following technologies:

Java: _____

JUnit: _____

Ant: _____

3. Has a build status notification device been used for a project you have worked on?

Yes

No

If yes, please describe _____

Appendix C: Second Evaluation Explanation and Guidelines – Part I

Continuous Integration Notification User Study

Part I

Experiment Facilitator: Ruth M. Ablett
ICT 524, University of Calgary

Thank you for your participation in this study. Your participation is totally voluntary and you can withdraw from this study at any time.

Experiment Summary

Everyone involved in this study is working to develop a different aspect of a shared project, AgilePlanner. To make sure all the parts work together correctly, continuous integration will be used. So when any developer commits code to the repository, the code will be compiled together, deployed, and a series of tests will be executed. If the state of the build changes (e.g. if it changes from success to failure, or from failure to success), you will be alerted via email.

When you commit code, an email will be sent to you with an update of the build status. If you committed code that caused the build to break, the email will contain information about the breakage including which tests failed. This email will also be sent to the project manager (Dr. Frank Maurer) and the experiment facilitator (myself).

We will need a working email address from each participant (if you do not wish to give out your email, a temporary free email account can be set up for you for the duration of the study).

The Commit Mouse. There is a stuffed red 'commit mouse' in the lab. Its purpose is to restrict who is able to commit the code. You may only commit if the red mouse is at your desk, and you may only give it to someone else if the code you have committed results in an intact build. If you are not in the lab, please check the build via <http://zeus.cpsc.ucalgary.ca:8080/anthill/> to make sure the build is intact before committing your code.

Do not be afraid to commit new code! It is important that new code is committed often enough to validate this study. As an incentive, we are offering a \$10 University of Calgary Micro Store gift voucher to the developer who makes the highest number of unique commits this week. Also, we are offering a free pizza lunch at the end of the study for all participants.

Specifics

Schedule. This study will take place on weekdays between 9:00am to 5:00pm.

During this time you are free to take any breaks you require, but please try to be in the laboratory as much as possible during these hours.

Location. This study will take place in the north half of the ICT 524 lab.

Participants. The participants will be a combination of graduate students working part-time and interns working full-time on the project together.

Feel free to collaborate with other participants (pair programming, etc).

You may drop from this study at any time without penalty and your results will not be used.

Appendix D: Second Evaluation Post-Questionnaire – Part I (Developers)

1. Did you use an email notification program (ie. Google Talk, MSN messenger, etc)?

Yes

No

2. How were you **most effectively** notified to a build problem?

a) I received an email

b) I checked the build status page at <http://zeus.cpsc.ucalgary.ca:8080/anthill/>

e) I was alerted by another person

Please answer the following questions with a scale of 1 to 5, where 1 = *strongly disagree* and 5 = *strongly agree*.

The email was a distraction.

1

2

3

4

5

My ability to perceive things going on around me was affected (eg. headphones)

1

2

3

4

5

I felt I was aware of the build status at all times.

1

2

3

4

5

Appendix E: Second Evaluation Post-Questionnaire – Part I (Observers)

How were you **most effectively** notified re: the build status?

- a) I checked the build status page at <http://zeus.cpsc.ucalgary.ca:8080/anthill/>
- b) I was alerted by another person

Please answer the following questions with a scale of 1 to 5, where 1 = *strongly disagree* and 5 = *strongly agree*.

My ability to perceive things going on around me was affected (eg. headphones)

1 2 3 4 5

I felt I was aware of the build status at all times.

1 2 3 4 5

Appendix F: Second Evaluation Explanation and Guidelines – Part II

Continuous Integration Notification User Study Part II

Experiment Facilitator: Ruth M. Ablett
ICT 524, University of Calgary

Thank you for your participation in this study. Your participation is totally voluntary and you can withdraw from this study at any time.

Experiment Summary

Everyone involved in this study is working to develop a different aspect of a shared project, AgilePlanner. To make sure all the parts work together correctly, continuous integration will be used. So when any developer commits code to the repository, the code will be compiled together, deployed, and a series of tests will be executed. If the state of the build changes (e.g. if it changes from success to failure, or from failure to success), you will be alerted via email, as well as a change in the lava lamps situated in the center of the lab.

As before, when you commit code, an email will be sent to you with an update of the build status. If you committed code that caused the build to break, the email will contain information about the breakage including which tests failed. This email will also be sent to the project manager (Dr. Frank Maurer) and the experiment facilitator (myself).

In addition, lava lamps will be used to convey the state of the build. There are two lava lamps, one red and one green, on top of one of the cabinets in the experiment space. These lamps describe the state of the build – only one of them can be on at any given time. If the green lamp is on (and the red lamp is off), the last build was successful and all tests passed. If the red lamp is on (and the green lamp is off), it means the last build failed and it must be repaired.

Please familiarize yourself with the location of the lamps from where you are sitting.

When this occurs, a soft but audible click can be heard. This click is the sound of the receiver unit controlling power to the lamps. Similarly, if the build is fixed, the red lamp will turn off, and the green lamp will be turned on (lower right). This will also be accompanied by the click.

The lamps can become quite hot during operation, so please do not touch them.

The Commit Mouse. There is a stuffed red 'commit mouse' in the lab. Its purpose is to restrict who is able to commit the code. You may only commit if the red mouse is at your desk, and you may only give it to someone else if the code you have committed results in an intact build. If you are not in the lab, please check the build via <http://zeus.cpsc.ucalgary.ca:8080/anthill/> to make sure the build is intact before committing your code.



Do not be afraid to commit new code! It is important that new code is committed often enough to validate this study. As an incentive, we are offering a \$10 University of Calgary Micro Store gift voucher to the developer who makes the highest number of unique commits this week. Also, we are offering a free pizza lunch at the end of the study for all participants.

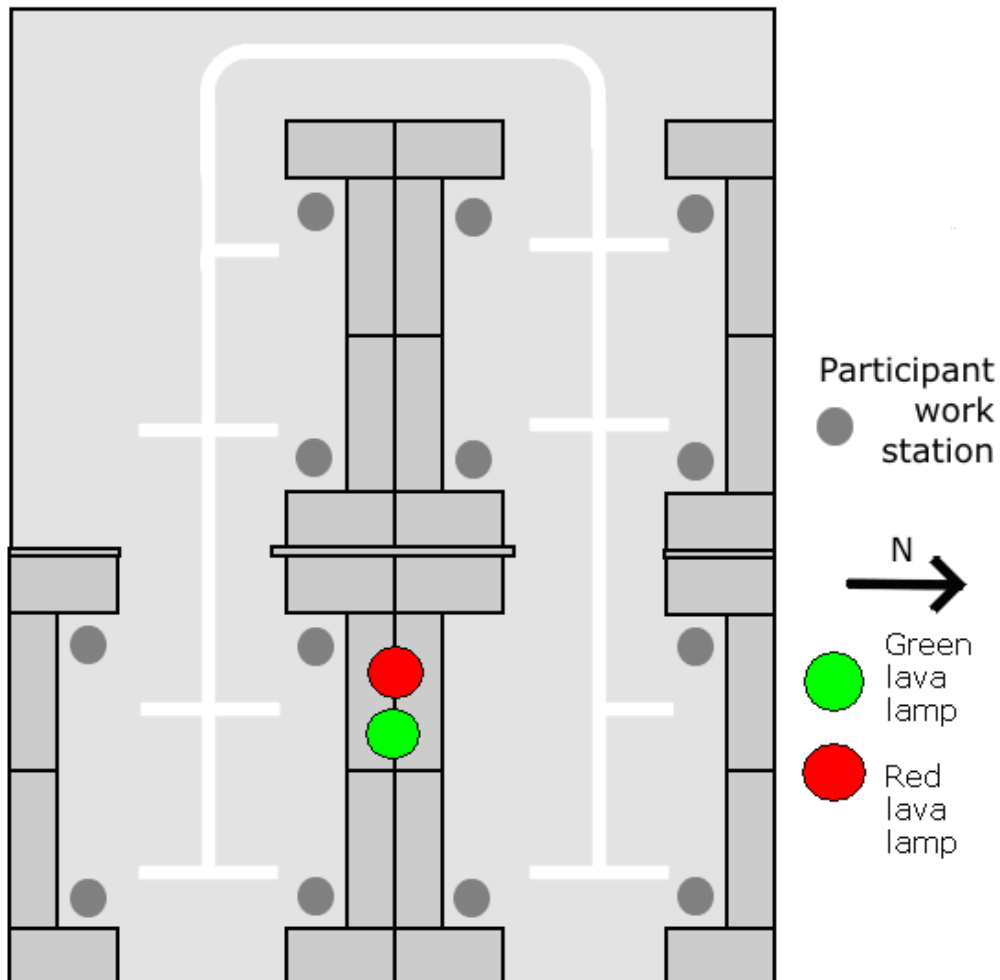
Specifics

Schedule. This study will take place on weekdays between 9:00am to 5:00pm.

During this time you are free to take any breaks you require, but please try to be in the laboratory as much as possible during these hours.

Location. This study will take place in the north half of the ICT 524 lab.

The following is a map of the laboratory where the experiment will take place.



Participants. The participants will be a combination of graduate students working part-time and interns working full-time on the project together.

Feel free to collaborate with other participants (pair programming, etc).

You may drop from this study at any time without penalty and your results will not be used.

Appendix G: Second Evaluation Post-Questionnaire – Part II (Developers)

1. Did you use an email notification program (ie. Google Talk, MSN messenger, etc)?

Yes

No

2. How were you **most effectively** notified to a build problem?

a) I received an email

b) I saw the lava lamps change

c) I noticed the lava lamp bubbles moving

d) I heard the click of the lava lamp controller

e) I was alerted by another person

f) I checked the build status page at <http://zeus.cpsc.ucalgary.ca:8080/anthill/>

Please answer the following questions with a scale of 1 to 5, where 1 = *strongly disagree* and 5 = *strongly agree*.

The sight of the lava lamps was a distraction.

1 2 3 4 5

The clicking sound of the lava lamps was a distraction.

1 2 3 4 5

The email was a distraction.

1 2 3 4 5

The lava lamps contributed to a fun atmosphere.

1 2 3 4 5

I have a clear view of the lava lamps from my desk.

Blocked

1

2

3

4

In clear view

5

My ability to perceive things going on around me was affected (eg. headphones)

1 2 3 4 5

I felt I was aware of the build status at all times.

1 2 3 4 5

Appendix H: Second Evaluation Post-Questionnaire – Part II (Observers)

How were you **most effectively** notified to a build problem?

- a) I saw the lava lamps change
- b) I noticed the lava lamp bubbles moving
- c) I heard the click of the lava lamp controller
- d) I was alerted by another person
- e) I checked the build status page at <http://zeus.cpsc.ucalgary.ca:8080/anthill/>

Please answer the following questions with a scale of 1 to 5, where 1 = *strongly disagree* and 5 = *strongly agree*.

The sight of the lava lamps was a distraction.

1 2 3 4 5

The clicking sound of the lava lamps was a distraction.

1 2 3 4 5

The lava lamps contributed to a fun atmosphere.

1 2 3 4 5

I have a clear view of the lava lamps from my desk.

Blocked 1 2 3 4 **In clear view**
1 2 3 4 5

My ability to perceive things going on around me was affected (eg. headphones)

1 2 3 4 5

I felt I was aware of the build status at all times.

1 2 3 4 5

Appendix I: Second Evaluation Explanation and Guidelines – Part III

Continuous Integration Notification User Study

Part III

Experiment Facilitator: Ruth M. Ablett
ICT 524, University of Calgary

Thank you for your participation in this study. Your participation is totally voluntary and you can withdraw from this study at any time.

Experiment Summary

Everyone involved in this study is working to develop a different aspect of a shared project, AgilePlanner. To make sure all the parts work together correctly, continuous integration will be used. So when any developer commits code to the repository, the code will be compiled together, deployed, and a series of tests will be executed. If the state of the build changes (e.g. if it changes from success to failure, or from failure to success), you will be alerted via email, as well as the deployment of BuildBot, a software development monitor robot.

As before, when you commit code, an email will be sent to you with an update of the build status. If you committed code that caused the build to break, the email will contain information about the breakage including which tests failed. This email will also be sent to the project manager (Dr. Frank Maurer) and the experiment facilitator (myself).

BuildBot is a white Sony AIBO ERS-7M2. If a code check-in by a developer results in a state change to build failure, BuildBot will make its way to that developer's desk. If the build is fixed, the robot will turn around (even halfway to the desk) and return to its base station.

The network of white lines on the floor is the map BuildBot uses to get to each desk. Please try not to leave obstacles on these lines. Also, if you need to leave your workstation and you hear the robot moving, please be careful not to trip over it.

The Commit Mouse. There is a stuffed red 'commit mouse' in the lab. Its purpose is to restrict who is able to commit the code. You may only commit if the red mouse is at your desk, and you may only give it to someone else if the code you have committed results in an intact build. If you are not in the lab, please check the build via <http://zeus.cpsc.ucalgary.ca:8080/anthill/> to make sure the build is intact before committing your code.

Do not be afraid to commit new code! It is important that new code is committed often enough to validate this study. As an incentive, we are offering a \$10 University of Calgary Micro Store gift voucher to the developer who makes the highest number of unique commits this week. Also, we are offering a free pizza lunch at the end of the study for all participants.

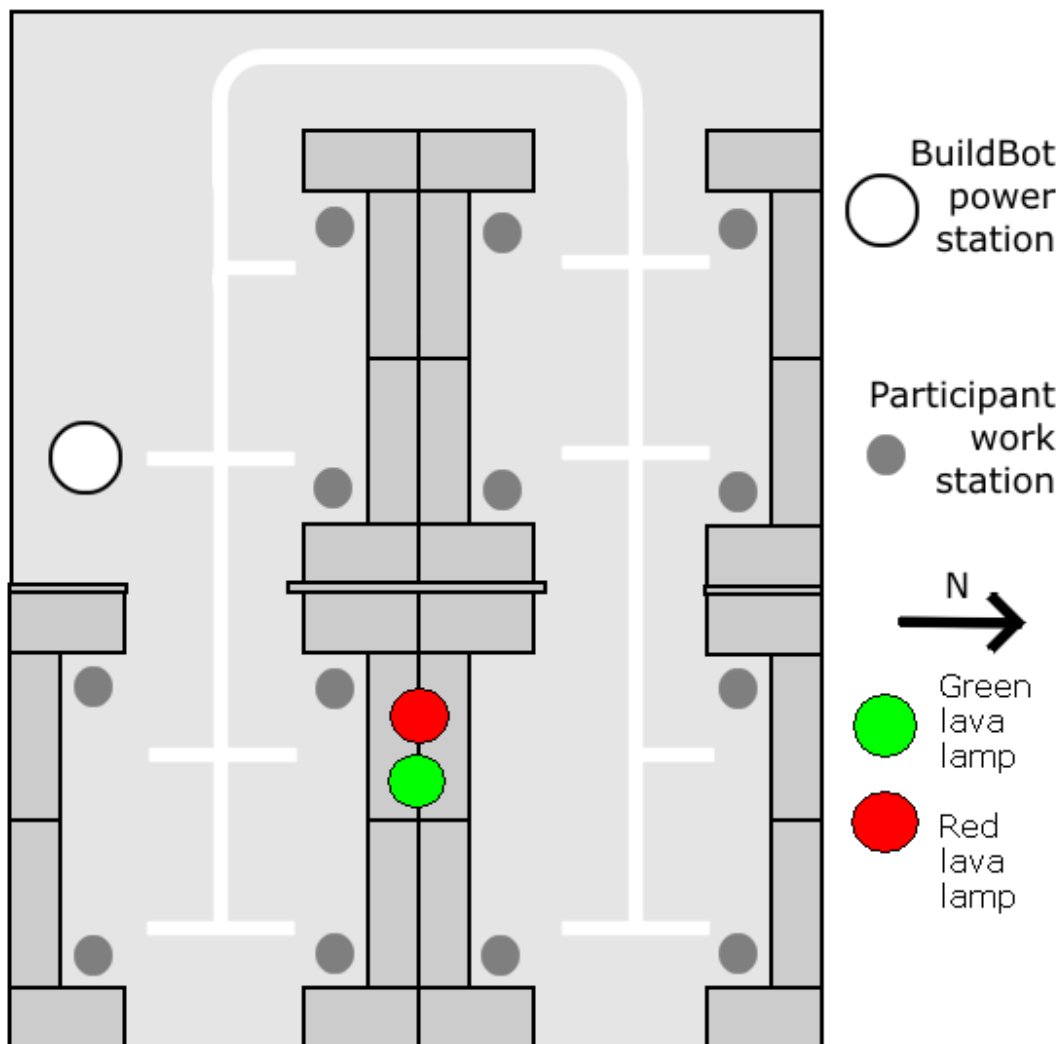
Specifics

Schedule. This study will span a total of one week, on weekdays between Monday, 21 May and Monday, 28 May. The study will run for eight hours each day, from 9:00am to 5:00pm daily.

During this time you are free to take any breaks you require, but please try to be in the laboratory as much as possible during these hours.

Location. This study will take place in the north half of the ICT 524 lab.

The following is a map of the laboratory where the experiment will take place.



Participants. The participants will be a combination of graduate students working part-time and interns working full-time on the project together.

Feel free to collaborate with other participants (pair programming, etc).

You may drop from this study at any time without penalty and your results will not be used.

Appendix J: Second Evaluation Post-Questionnaire – Part III (Developers)

4. Did you use an email notification program (ie. Google Talk, MSN messenger, etc)?

Yes

No

5. How were you **most effectively** notified to a build problem?

a) I received an email

b) I saw the robot moving

c) I heard the sound of the robot's joints moving

d) I heard non-joint noises from the robot (ie. growling)

e) The level of activity in the lab changed

f) I was alerted by another person who saw or heard the robot

g) I checked the build status page at <http://zeus.cpsc.ucalgary.ca:8080/anthill/>

h) I was alerted by another person who checked the build status page

i) I was alerted by another person who received an email

Please answer the following questions with a scale of 1 to 5, where 1 = *strongly disagree* and 5 = *strongly agree*.

The sight of the robot moving was a distraction.

1

2

3

4

5

The sound of the robot moving was a distraction.

1

2

3

4

5

The email was a distraction.

1

2

3

4

5

The robot contributed to a fun atmosphere.

1

2

3

4

5

My ability to perceive things going on around me was affected (eg. headphones)

1 2 3 4 5

I felt I was aware of the build status at all times.

1 2 3 4 5

Which notification device did you prefer?

Email alone

Lava Lamps

BuildBot

Please explain:

Appendix K: Second Evaluation Post-Questionnaire – Part III (Observers)

6. Did you use an email notification program (ie. Google Talk, MSN messenger, etc)?

Yes

No

7. How were you **most effectively** notified to a build problem?

a) I saw the robot moving

b) I heard the sound of the robot's joints moving

c) I heard non-joint noises from the robot (ie. growling)

d) The level of activity in the lab changed

e) I was alerted by another person who saw or heard the robot

f) I checked the build status page at <http://zeus.cpsc.ucalgary.ca:8080/anthill/>

g) I was alerted by another person who checked the build status page

h) I was alerted by another person who received an email

Please answer the following questions with a scale of 1 to 5, where 1 = *strongly disagree* and 5 = *strongly agree*.

The sight of the robot moving was a distraction.

1 2 3 4 5

The sounds (joints, growling, etc) made by the robot were a distraction.

1 2 3 4 5

The energy in the room was a distraction.

1 2 3 4 5

The robot contributed to a fun atmosphere.

1 2 3 4 5

My ability to perceive things going on around me was affected (eg. headphones)

1 2 3 4 5

I felt I was aware of the build status at all times.

1 2 3 4 5

Which notification device did you prefer?

Email alone

Lava Lamps

BuildBot

Please explain:

Appendix L: Participant Consent Form



Name of Researcher, Faculty, Department, Telephone & Email:

Ruth Ablett

Supervisor:

Dr. Frank Maurer, Dr. Ehud Sharlin

Title of Project:

ALAN: A robotic companion for agile teams

This consent form, a copy of which has been given to you, is only part of the process of informed consent. If you want more details about something mentioned here, or information not included here, you should feel free to ask. Please take the time to read this carefully and to understand any accompanying information.

The University of Calgary Conjoint Faculties Research Ethics Board has approved this research study.

Purpose of the Study:

You are invited to join a development team to participate in a study comparing different kinds of build notification alerts. If someone in the development team breaks the build, the team, as well as other individuals such as the project manager, must be alerted somehow. We will test several types of notifications.

What Will I Be Asked To Do?

You will be a part of a team of developers working on a software project called AgilePlanner. While programming, bugs may be introduced to the system. When this occurs, the team will be alerted to the build failure via several different methods. You will be asked to comment on each type of alert and describe its effect on you as well as the whole team.

This study will test several of these build notification techniques.

What Type of Personal Information Will Be Collected?

Should you agree to participate, you will be asked to provide some details about your experience with computer programming in general.

There are several options for you to consider if you decide to take part in this research. You can choose all, some or none of them. Please put a check mark on the corresponding line(s) that grants me your permission to:

I grant permission to be audio taped: Yes: ___ No: ___

I grant permission to be videotaped: Yes: ___ No: ___

I wish to remain anonymous: Yes: ___ No: ___

I wish to remain anonymous, but you may refer to me by a pseudonym: Yes: ___ No: ___

The pseudonym I choose for myself is: _____

You may quote me and use my name: Yes: ___ No: ___

Please note that should you grant us permission to capture and use your image presentations/reports, your anonymity will be compromised.

Are there Risks or Benefits if I Participate?

There are no known risks, however, if you feel uncomfortable you are free to quit at any time. The data collected to the point of withdrawal will remain in the system for the researcher's analysis.

There will be a free pizza lunch at the end of the study for all participants.

What Happens to the Information I Provide?

Collection Information:

Participation is completely voluntary. You are free to discontinue participation at any time during the study. We will get your explicit consent if we use the raw data for presentation purposes. The raw data collected will be kept in a secure server where accessible only by the researcher and the supervisor. The data will be stored for two years on a computer disk, at which time, it will be permanently erased. If you withdraw from the participation, the data collected to the point of withdrawal will be retained / used.

Signatures (written consent)

Your signature on this form indicates that you 1) understand to your satisfaction the information provided to you about your participation in this research project, and 2) agree to participate as a research subject.

In no way does this waive your legal rights nor release the investigators, sponsors, or involved institutions from their legal and professional responsibilities. You are free to withdraw from this research project at any time. You should feel free to ask for clarification or new information throughout your participation.

Participant's Name: (please print) _____

Participant's Signature _____ Date: _____

Researcher's Name: (please print) _____

Researcher's Signature: _____ Date: _____

Questions/Concerns

If you have any further questions or want clarification regarding this research and/or your participation, please contact:

*Ruth Ablett
Department of Computer Science / Faculty of Graduate Studies
403.220.7140 / abletr@cpsc.ucalgary.ca
And
Dr. Frank Maurer, Department of Computer Science
Dr. Ehud Sharlin, Department of Computer Science*

If you have any concerns about the way you've been treated as a participant, please contact Bonnie Scherrer, Ethics Resource Officer, Research Services Office, University of Calgary at (403) 220-3782; email bonnie.scherrer@ucalgary.ca.

A copy of this consent form has been given to you to keep for your records and reference. The investigator has kept a copy of the consent form.

Appendix M: Ethics and Co-Author Approval



UNIVERSITY OF
CALGARY

MEMO

CONJOINT FACULTIES RESEARCH ETHICS BOARD
c/o Research Services
Main Floor, Energy Resources Research Building
3512 - 33 Street N.W., Calgary, Alberta T2L 1Y7
Telephone: (403) 220-3782
Fax: (403) 289 0693
Email: bonnie.scherrer@ucalgary.ca
Wednesday, February 01, 2006

To: Shelly S. Park
Computer Science

From: Dr. Janice P. Dickin, Chair
Conjoint Faculties Research Ethics Board (CFREB)

Re: Certification of Institutional Ethics Review: ALAN: A Robotic Companion for Agile Teams

The above named research protocol has been granted ethical approval by the Conjoint Faculties Research Ethics Board for the University of Calgary.

Enclosed are the original, and one copy, of a signed **Certification of Institutional Ethics Review**. Please make note of the conditions stated on the Certification. A copy has been sent to your supervisor as well as to the Chair of your Department/Faculty Research Ethics Committee. In the event the research is funded, you should notify the sponsor of the research and provide them with a copy for their records. The Conjoint Faculties Research Ethics Board will retain a copy of the clearance on your file.

Please note, an annual/progress/final report must be filed with the CFREB twelve months from the date on your ethics clearance. A form for this purpose has been created, and may be found on the "Ethics" website, <http://www.ucalgary.ca/UofC/research/html/ethics/reports.html>

In closing let me take this opportunity to wish you the best of luck in your research endeavor.

Sincerely,

A handwritten signature in black ink, appearing to read 'B. Scherrer'.

Bonnie Scherrer

For:

Janice Dickin, Ph.D., LL.B., Faculty of Communication and Culture and
Chair, Conjoint Faculties Research Ethics Board

Enclosures(2)

cc: Chair, Department/Faculty Research Ethics Committee
Supervisor: Frank Maurer



CERTIFICATION OF INSTITUTIONAL ETHICS REVIEW

This is to certify that the Conjoint Faculties Research Ethics Board at the University of Calgary has examined the following research proposal and found the proposed research involving human subjects to be in accordance with University of Calgary Guidelines and the Tri-Council Policy Statement on "Ethical Conduct in Research Using Human Subjects". This form and accompanying letter constitute the Certification of Institutional Ethics Review.

File no: 4689
Applicant(s): Shelly S. Park, Ruth M. Ablett, Frank Maurer, Joerg Denzinger, Ehud Sharlin
Department: Computer Science
Project Title: ALAN: A Robotic Companion for Agile Teams
Sponsor (if applicable):

Restrictions:

This Certification is subject to the following conditions:

- 1. Approval is granted only for the project and purposes described in the application.
2. Any modifications to the authorized protocol must be submitted to the Chair, Conjoint Faculties Research Ethics Board for approval.
3. A progress report must be submitted 12 months from the date of this Certification, and should provide the expected completion date for the project.
4. Written notification must be sent to the Board when the project is complete or terminated.

Signature of Janice Dickin, Ph.D., LL.B., Chair, Conjoint Faculties Research Ethics Board

Date: 1 February 2006

Distribution: (1) Applicant, (2) Supervisor (if applicable), (3) Chair, Department/Faculty Research Ethics Committee, (4) Sponsor, (5) Conjoint Faculties Research Ethics Board (6) Research Services.



Conjoint Faculties Research Ethics Board (CFREB)
Research Services Office
Main Floor, Energy Resources Research Building
Research Park
Telephone: (403) 220-3782
Fax: (403) 289-0693
Email: bonnie.scherrer@ucalgary.ca

To: Ruth Ablett
Department of Computer Science

Date: May 4, 2007

From: Dr. J. Kent Donlevy, Acting Chair
Conjoint Faculties Research Ethics Board

Re: Approval of Modification for: ALAN: A Robotic Companion for Agile Teams
Original Approval Date: 01 February 2006
File No: 4689

The Certificate of Institutional Ethics Review issued on 01 February 2006 continues in force and extends to the modifications as set out in your e-mails and attachments dated 02 and 03 May 2007. Your request to run a "scaled-back" phase of the study where participants will be asked to evaluate various alert/notification techniques for bugs introduced into the system during a software project, using the revised consent form submitted with your 03 May e-mail, is approved, as described.

You should attach a copy of the documentation you provided in order to request the modifications, together with a copy of this memorandum, to the original Certification in your files.

Sincerely,

J. Kent Donlevy, LLB, PhD, Associate Professor
Faculty of Education
Acting Chair, Conjoint Faculties Research Ethics Board

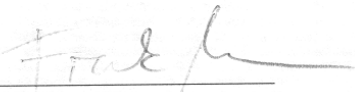
Cc: S. Park
J. Denziger (Co-Investigators)
F. Maurer
E. Sharlin (Co-Investigators/Supervisors)

May, 2007

University of Calgary
2500 University Drive NW
Calgary, Alberta
T2N 1N4

I, Frank Maurer, give Ruth Ablett permission to use co-authored work from our published paper, "BuildBot: A Robotic Self-Supervision Mechanism for Agile Software Engineering Teams" in this thesis.

Sincerely,



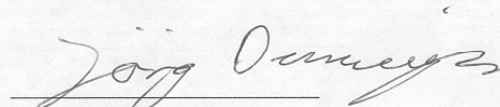
Frank Maurer

May, 2007

University of Calgary
2500 University Drive NW
Calgary, Alberta
T2N 1N4

I, Joerg Denzinger, give Ruth Ablett permission to use co-authored work from our published paper, "BuildBot: A Robotic Self-Supervision Mechanism for Agile Software Engineering Teams" in this thesis.

Sincerely,



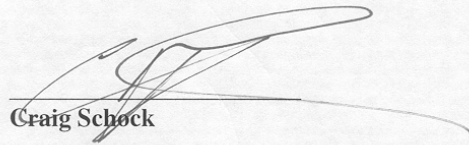
Joerg Denzinger

May, 2007

University of Calgary
2500 University Drive NW
Calgary, Alberta
T2N 1N4

I, Craig Schock, give Ruth Ablett permission to use co-authored work from our published paper, "BuildBot: A Robotic Self-Supervision Mechanism for Agile Software Engineering Teams" in this thesis.

Sincerely,



Craig Schock

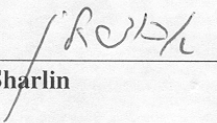
May 22-07

May, 2007

University of Calgary
2500 University Drive NW
Calgary, Alberta
T2N 1N4

I, Ehud Sharlin, give Ruth Ablett permission to use co-authored work from our published paper, "BuildBot: A Robotic Self-Supervision Mechanism for Agile Software Engineering Teams" in this thesis.

Sincerely,



Ehud Sharlin