

May 2015

# Expanding the Impact of the EEROS Open Source Robotics Framework

Nathan Harold Hughes  
*Worcester Polytechnic Institute*

Nicholas Francis Brown  
*Worcester Polytechnic Institute*

Nicholas Frederick Hassan  
*Worcester Polytechnic Institute*

Ryan Thomas Lang  
*Worcester Polytechnic Institute*

Follow this and additional works at: <https://digitalcommons.wpi.edu/iqp-all>

---

## Repository Citation

Hughes, N. H., Brown, N. F., Hassan, N. F., & Lang, R. T. (2015). *Expanding the Impact of the EEROS Open Source Robotics Framework*. Retrieved from <https://digitalcommons.wpi.edu/iqp-all/3033>

This Unrestricted is brought to you for free and open access by the Interactive Qualifying Projects at Digital WPI. It has been accepted for inclusion in Interactive Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact [digitalwpi@wpi.edu](mailto:digitalwpi@wpi.edu).

# Expanding the Impact of the EEROS Open Source Robotics Framework

An Interactive Qualifying Project  
submitted to the Faculty of  
WORCESTER POLYTECHNIC INSTITUTE  
in partial fulfilment of the requirements for the  
degree of Bachelor of Science

by  
Nicholas Brown  
Nicholas Hassan  
Nathan Hughes  
Ryan Lang

Date:  
1 May 2015

Report Submitted to:

Professor Einar Nielsen  
NTB Buchs

Professor Scott Justo  
Professor Ruth Smith  
Worcester Polytechnic Institute

*This report represents work of WPI undergraduate students submitted to the faculty as evidence of a degree requirement. WPI routinely publishes these reports on its web site without editorial or peer review. For more information about the projects program at WPI, see <http://www.wpi.edu/Academics/Projects>.*

## Abstract

This report, prepared for the developers of the EEROS Real Time Robotics Software Framework, explored options to expand the impact that EEROS would have on the open source robotics community. This open source framework was examined to discover how a healthy development community might grow in a new project. Through increasing EEROS's presence, analyzing its community, exploring sustainable funding options, organizing and streamlining development and identifying new partners, we gained an understanding of the birth of an open source project.

# Table of Contents

Abstract .....	ii
Table of Figures .....	v
Table of Tables .....	v
Chapter 1: Introduction .....	1
Chapter 2: Background .....	3
2.1 Free Software, the Open Source Movement and Open Source Communities .....	3
2.1.1 What Is the Open Source Movement? .....	3
2.1.2 History of the Open Source Movement .....	4
2.1.3 Ideologies of the Open Source Movement .....	5
2.1.4 Why the Open Source Movement Is Successful .....	6
2.1.5 Why People Contribute to the Open Source Movement .....	7
2.1.6 Security Concerns and Quality Control of Open Source Software .....	7
2.2 Creating and Managing Open Source Software .....	8
2.2.1 Tools and Community Involvement .....	8
2.2.2 Licenses .....	10
2.2.3 Ethics of Licenses .....	10
2.2.4 Examples of Licenses .....	11
2.3 What makes an Open Source Software Project Successful? .....	11
2.3.1 Case Study: Red Hat .....	11
2.3.2 Other Open Source Business Models .....	13
2.3.3 Crowdfunding and Open Source Software .....	15
2.4 The Robotics Community .....	15
2.4.1 The History of the Robot .....	16
2.4.2 Social Aspects of the Robotics Community .....	17
2.4.3 Differences Between Education and Industry .....	17
2.5 Open Source Robotics Frameworks .....	18
2.5.1 ROS .....	19
2.5.2 ROS-Industrial, Urbi, and The Player Project .....	20
2.5.3 The Orocos Project .....	21
2.6 EEROS .....	22
2.6.1 Introduction to EEROS .....	22
2.6.2 Problems EEROS Attempts to Solve .....	23
2.6.3 Current State of Development .....	24
2.7 Building a Community for EEROS .....	25
2.7.1 EEROS and the Robotics Industry .....	25
2.7.2 Potential Funding Options in Industry .....	26
2.7.3 EEROS and the World of Educational Robotics .....	27
2.7.4 Potential Funding Options in Education .....	27
Chapter 3: Methodology .....	30
3.1 Increasing EEROS Presence in the Robotics Community .....	31
3.1.1 EEROS Website .....	31
3.1.2 EEROS Wiki .....	32
3.1.3 Social Media .....	32

3.1.4 Other Materials .....	32
3.2 Providing access to Sustainable Funding.....	33
3.3 Organizing and Streamlining EEROS Development.....	34
3.4 Maintaining a Healthy Open Source Community.....	35
3.5 Identifying New Partners .....	36
Chapter 4: Results and Findings .....	38
4.1 Presence .....	38
4.1.1 EEROS Website.....	38
4.1.2 EEROS Wiki.....	40
4.1.3 Social Media .....	41
4.1.4 Wikipedia Page .....	41
4.1.5 Informational Brochure.....	42
4.2 Funding Proposal .....	42
4.3 Organizing and Streamlining EEROS Development.....	43
4.4 Maintaining a Healthy Open Source Community.....	44
4.5 Identifying New Partners .....	46
Chapter 5: Recommendations and Conclusions .....	47
5.1 EEROS Presence.....	47
5.2 Sustainable Funding.....	48
5.3 Streamlined Development.....	48
5.4 Healthy Community.....	49
5.5 New Partners .....	50
5.6 Conclusions.....	50
Bibliography .....	52
Appendix A: Interview Notes .....	56
Appendix B: Before and After Screenshots of EEROS Homepage .....	60
Before:.....	60
After:.....	61
Appendix C: Website Screenshots.....	62
Appendix D: Community Analysis Math .....	64
Appendix E: Development Resources Annotated Bibliography .....	67
Appendix F: Annotated List of Partners .....	69

## Table of Figures

Figure 1: A brief history of Red Hat.....	12
Figure 2: The three main components of Orocos.....	22
Figure 3: The relationship between blocks, signals, and time domains.....	24
Figure 4: Methodology Outline .....	30
Figure 5: Project Timeline .....	31
Figure 6: Funding Phases.....	34
Figure 7: Website Homepage.....	39
Figure 8: Visualization Tool .....	44
Figure 9: Old EEROS Homepage .....	60
Figure 10: New EEROS Homepage .....	61
Figure 11: “What is EEROS?” page .....	62
Figure 12: “Applications” page .....	62
Figure 13: “Get Involved” page.....	63
Figure 14: “The EEROS Team” page.....	63

## Table of Tables

Table 1: A summary of open source revenue models (Source: Modified from Rajala, 2007) .....	14
Table 2: Interview Notes.....	56
Table 3: Development Resources Annotated Bibliography.....	67
Table 4: Annotated List of Partners .....	69

## Chapter 1: Introduction

Writing software for robots is difficult. With such a large diversity of robots being developed, engineers and researchers spend a considerable amount of time rewriting old code for new robots. Currently, there is no way to avoid this waste of resources; most robotics software ends up being too specialized to transfer from robot to robot. As the field of robotics becomes increasingly complex and prevalent in everyday life, researchers and programmers will want to reuse established processes and algorithms. This means making sure these aspects of robotics are shared among the robotics community. Rather than be constructed from a stagnant point of view, sharing information allows a project to grow and adapt to new innovations and requirements.

A solution is currently in development by Professor Einar Nielsen at the *Interstaatliche Hochschule für Technik Neu-Technikum Buchs* (NTB), a Swiss systems engineering university. This solution is EEROS, the Easy, Elegant, Reliable, Open and Safe Real-Time Robotics Software Framework, which aims to be a common platform for robotics software development. EEROS is an operating system for robots, just as Windows and Mac are operating systems for a home computer. Windows can run the same programs on a variety of computers, while EEROS is designed to run the same algorithms on a variety of robots. This flexibility is a powerful tool that gives EEROS the potential to become an international standard. Unfortunately, EEROS is not yet mature enough for this scale of use. In this early stage, a project typically needs heavy community contribution and involvement to succeed. In projects like EEROS, this means acquiring funding and marketing resources to spark the creation of a passionate, self-sustaining community (Athey & Ellison, 2014).

To create this kind of community, EEROS follows the open source software development paradigm. In a technical sense, open source means the software's source code is available for anyone to read and modify. Over its four decade history, open source has matured from a development model into an ideology that promotes user contribution. Especially successful open source projects have a devoted community of users and developers. For example, Linux, a successful open source alternative to Windows or Mac, was developed entirely by the community that uses it. Like most open source software, Linux is distributed free of charge, but companies like Red Hat have implemented creative ways of generating revenue from a free product (Young, 1999).

EEROS is not the first robotics software framework. It has many competitors, several of which are already popular and successful. ROS, the Robotics Operating System, is one of the most successful open source robotics software frameworks. It draws that success from the software's flexibility, multitude of contributors, extensive documentation and devoted community (ROS, 2015). Other projects, such as Orocos and the Player Project, have found success for similar reasons. EEROS aims to solve several problems that have not been fully addressed by its competitors. While most other robotics frameworks depend on Linux to

function, EEROS is completely free standing. It also prioritizes safety, which is important for industrial applications. Finally, it offers real-time support, which has not been integrated into most robotics software frameworks. These features offer solutions to current issues in robotics today; this is why EEROS has the potential to find a place in the robotics community beside these other software frameworks. EEROS is also developed in an university where students and professors frequently build new robots, which has created an ethos that the framework needs to become flexible enough to handle new areas of robotics that have yet to be created.

EEROS is at a critical stage; it is transitioning from a local project to a community developed open source framework. As of now, the EEROS community consists of the core developers and a handful of users, all at NTB. To change this, effective marketing needs to be used to grow the community. Additionally, most open source projects are exclusively software based, but EEROS depends on hardware as well, which requires a reliable funding source. This dependency on hardware also creates a disconnect between contributors and users. Software developers are not necessarily roboticists, and vice versa. This leaves EEROS with the problem of attracting people who will use the software and be able to contribute to it. Also, reliability and safety, two of the essential attributes of EEROS, may conflict with keeping the software open. A system will have to be established to ensure the quality and safety of code that is shared among end users. All these problems challenge the potential for EEROS to reach its own vision of success.

The goal of this project is to work with the researchers, students, and professors currently developing the EEROS Open Source Robotics Software Framework to expand the impact it will have on the robotics community, and to understand how a developing open source project can transition to a successful project. Specifically, we aimed to work with the EEROS team to increase EEROS's presence in the robotics community, to analyze the development community health, to help the team gain access to sustainable funding, to organize and streamline EEROS development, and to identify new partners for expansion. Completing these objectives allowed us and the EEROS team to understand the challenges that face a burgeoning open source project and how a project can eventually transition to a global standard.

## Chapter 2: Background

The motivation for the EEROS open source robotics framework is to provide a common software platform for the robotics community that will foster a culture of collaboration and openness between community members. This provides several questions to explore: What is the robotics community? Why is a common platform for programming robots useful to the robotics community, and why hasn't it been done before? What does open source mean? Why would someone make a project open source? How can EEROS draw upon the successes of previous open source projects? These are all important questions, and we address them in this section of the paper. We first explore the concept of open source software, then move on to examining successful open source projects to give some perspective into how EEROS was developed. Finally, we explore the robotics community and how EEROS aims to fulfill some of their current needs.

### 2.1 Free Software, the Open Source Movement and Open Source Communities

The idea of open source is a driving force behind how software is developed. So what exactly does open source mean? To illustrate this, we start by explaining the history of the open source movement, then move on to the ideologies of the movement itself. We then examine why open source projects are successful, and the community behind them. Finally, we address some concerns of the open source movement, particularly with the quality and security of open source software.

#### 2.1.1 What Is the Open Source Movement?

The easy way to describe the open source movement is to say it is a group of programmers who began writing open source software. In reality, the situation is far more complex than that, and we start with the question: What exactly is open source software? To the layperson, the difference seems very much like the difference between generic and brand-name food. Open source software is usually inexpensive or free, but is much less exciting to use. When you're looking for software to fulfill functionality, such as a word processor, most users will pick the more established, more marketed version, like Microsoft Word. The same experience could be had for free by using LibreOffice Writer, a prominent open source alternative. When looking for new software, many users do not consider open source as enough of a benefit to switch (Carillo & Okoli, 2008).

This is because the typical user doesn't have any use for the basic distinction between open source software and proprietary software (Bonaccorsi & Rossi, 2003). The distinction is that the source code for the software is open, or available for public viewing. If Microsoft Word was open source, any user could take the code and reconstruct Microsoft Word on their own computer. In addition, any user could then make modifications to the code. If you didn't like how Word does their toolbars, you could write your own toolbar system. And if you thought

your version of Word was better, you could release it under a new name for more people to download and use.

Not all open source software allows modifications and redistribution of source code. We go over the distinctions between the different permissions users have later in detail, but these different permissions arose from the how the open source movement started in the first place.

### 2.1.2 History of the Open Source Movement

Even though the open source movement is simple to define, the history of the movement is a little more unique. Open source software comes with a lot of ethical dilemmas, and most of them caused schisms in the community (Grinzo & Fernandez, 1999). This is impressive given how recent the movement started, but unsurprising given how ubiquitous and relevant the ideas are. In this section, we look at three major figures of the open source movement that each marked the arrival of a significant faction of the open source movement.

The first of these is Richard Stallman. His contribution to the open source movement began with a printing problem; and as a programmer, his first solution was to change some of the printer code. Unfortunately, the code that controlled the printer wasn't available for him to change (Williams, 2004). This is one of the typical reasons an open source software project starts and the real purpose of open source software comes into play: An end user runs into a limitation of an original software project, and wants to make a change to continue using the software (Bonaccorsi & Rossi, 2003). Stallman recognized this need and founded the Free Software Foundation (FSF) in 1985. The FSF is dedicated to making sure that software allows users to view, modify, and redistribute source code at will (Carillo & Okoli, 2008). Since then, the FSF has made several significant contributions to the open source movement, primarily by writing the GNU Public License (abbreviated as GPL) (Carillo & Okoli, 2008), the importance of which will be addressed when we discuss licenses later in this paper. Richard Stallman championed the right for users to freely view, modify, and redistribute source code.

The next figure is Linus Torvalds. Torvalds didn't so much change the ideology of the open source movement as make it successful. Torvalds started two major open source projects: Linux, and git (Carillo & Okoli, 2008). These two projects were very different from a typical project, as Eric Raymond puts it:

Linus Torvalds's style of development—release early and often, delegate everything you can, be open to the point of promiscuity—came as a surprise. No quiet, reverent cathedral-building here—rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches (aptly symbolized by the Linux archive sites, who'd take submissions from *anyone*) out of which a coherent and stable system could seemingly emerge only by a succession of miracles. (Raymond, 1999)

Torvalds began a community-based development model; i.e. he realized Richard Stallman's vision that programmers would be able to freely contribute and advance the development of software that they use. Linux and Git, Torvalds's most significant projects, were tools to enable

open source development. Linux is an open source operating system (like Windows or OSX), and Git is a way to manage changes to source code from multiple people. These two tools served as a platform for open source developers to begin collaborating (Fitzpatrick & Collins-Sussman, 2012).

The third figure, Eric Raymond, marked a shift in ideology. Raymond realized that community based development (the Torvalds model of writing software) was an extremely powerful tool in creating software, but most companies were thrown by Stallman's notion of "free" software (Carillo & Okoli, 2008). He, along with another member, Brian Peren, coined the term open source software to try and alleviate some of those fears (Carillo & Okoli, 2008). Raymond's argument is best summed up by:

Perhaps in the end the open source culture will triumph not because cooperation is morally right or software "hoarding" is morally wrong (assuming you believe the latter, which neither Linus nor I do), but simply because the closed-source world cannot win an evolutionary arms race with open source communities that can put orders of magnitude more skilled time into a problem. (Raymond, 1999)

Raymond's major contribution was that he turned the ideology and social activism of Stallman and the FSF into an effective business and development strategy.

The history of the open source movement highlights how it grew from meager beginnings in a lab at MIT (Williams, 2004) into a worldwide movement with many successful projects to its name. The thing to remember is the relatively short time span in which it developed, and how quickly it became a huge phenomenon. Stallman founded the FSF in 1985, Torvalds started Linux in 1991, and Raymond released his essay, "The Cathedral and the Bazaar" in 1998 (Carillo & Okoli, 2008) (Raymond, 1999). The continued activism of these three figures, along with many others, has led to very different visions of the open source movement, and a very relevant force in software development.

### 2.1.3 Ideologies of the Open Source Movement

The different visions of the open source movement boil down into two categories: free software, and open source software. While these two visions were discussed briefly in the section before, it is important to understand how exactly they differ; they play a large role in the type of community formed around a project.

Free software is software that protects a user's specific rights. To paraphrase Stallman, free software gives the user the ability to run the software, obtain the source code, modify the source code, and release the source code (Stallman, 2009). Free software is a bit of a misnomer; as Stallman puts it, "Think of 'free speech,' not 'free beer'" (Stallman, 2009). To avoid confusion, the term *gratis* is used for software that is free of cost, while *libre* is used to mean free speech. For our purposes, we will use free when referring to *gratis* software, and open source when referring to *libre* software. Stallman's intended purpose of making software "free" isn't

commercially seated, but rather is concerned with preserving basic liberties of the user. This leads to some conflict with open source software.

Open source software shares many of the hallmarks of free software, and often, open source software can also be considered free software (Stallman, 2009). However, open source is more appropriately defined as a development model, rather than an ideology. By making software open source, you invite contributions from a community surrounding that software, but companies can still limit how it is distributed and contributed to.

#### **2.1.4 Why the Open Source Movement Is Successful**

Open source software is a very foreign concept to a typical business. Raymond claims that most traditional software development managers feel that a very organized, very structured product is the only way to get work done (Raymond, 1999). Bonaccorsi and Rossi (2003), claim that “at first sight [the open source movement] would seem like nonsense.” It seems hard to coordinate the hundreds of people who get involved in a project without any defined hierarchy. Furthermore, contributors come and leave as they please, leaving work unfinished. Finally, there is no clear sense of direction to the project, because there is no upper management to define what the project is intended to accomplish. These issues make trying to develop open source software seem like a “nonsense” idea.

Many of these problems are simply a disconnect between traditional management styles and open source development. Many of the reasons we just provided seem valid, when trying to analyze open source development from a traditional context. But open source development is quite different than traditional management of projects, and trying to approach it in the same way is a mistake. These three reasons described above actually play a huge part in why open source development is successful.

To coordinate hundreds of developers, open source structures create hierarchy on the fly, depending on the needs of the project (Bonaccorsi & Rossi, 2003). This is why there is such a disconnect between traditional projects and open source projects. A manager in a traditional project is assigned at the beginning of the project, and carries out the same role for the duration of the project (Bonaccorsi & Rossi, 2003). Instead, a higher up in open source software is responsible for managing a component that they are knowledgeable about, by selecting appropriate solutions to problems provided by lower level contributors (Bonaccorsi & Rossi, 2003). After that certain component is developed, there is no longer a need for that higher-up, and the position dissolves.

What about the other problems with open source? If contributors don’t ever finish developing a critical component of the project before leaving, the project isn’t going to move forward. At least in a traditional software project, the managers can assign people to focus on those areas to make sure that the project can move forward. The benefit of open source software in this case is that if the project does end up stalling, contributors will know to focus on that area to make the project succeed (Bonaccorsi & Rossi, 2003). This, like the dynamic hierarchical

structure, offers the open source community a unique advantage of being flexible and responsive to the problem being solved. This flexibility is the reason why open source software projects tend to be so successful.

### **2.1.5 Why People Contribute to the Open Source Movement**

We've explored the generalities of open source software so far: how it arose, the ethos of the different factions of the open source movement, and why it's been so successful. But how do you start a piece of open source software? How do you get people to contribute to your project? These are two of the questions that the EEROS developers should be asking. The answer usually lies in forming a community around some need or problem. But forming a community isn't always easy. There's not a lot of external motivation for contributing to an open source project, in some cases, you won't be paid for it, and you're not any more likely to get famous from an open source project than a proprietary project. As Glass (1999) says, "I don't know who these crazy people are who want to write, read and even revise all that code without being paid anything for it at all." So why do people contribute?

Bonaccorsi and Rossi (2003) outlines three intrinsic reasons why programmers contribute to an open source project: there's a sense of satisfaction from solving a complex problem, a sense of pride over how elegant a particular solution is, and a sense of creative freedom not found in proprietary software. Lerner and Tirole (2001) states that contributors do so for recognition from their peers in the community. Contribution itself motivates other community members to contribute. By contributing some fixes, members of the project depend on the fact that another member, later down the road, will come up with a fix they need (Lerner & Tirole, 2001), which is considered reciprocal altruism (Athey & Ellison, 2014).

All of these reasons tend to build stronger, more skilled communities around software projects. Raymond (1999) states that open source projects usually attract the top 5% of programmers, and that these top programmers can contribute about 100 times as much as the less skilled programmers. Extraordinary communities develop from skilled programmers with a powerful motivation to contribute to an open source project. To build this kind of community around a project, you have to try to create a culture that promotes these motivations. This is something that EEROS needs to tap into to be successful.

### **2.1.6 Security Concerns and Quality Control of Open Source Software**

A lot of what we've discussed on open source software relies on contributors being motivated, competent, and altruistic. But what happens when a contributor makes a mistake in his or her part of the code? Or what if a contributor maliciously adds security flaws? These two concerns are at the heart of any open source project.

The first problem is dealt with by a quality assurance process. Different open source projects have different mechanisms to do this, but they primarily rely on two things: a mechanism to report and track bugs (errors in the code that cause undesirable behavior for the user) that the project has produced, and a specified procedure for dealing with bugs (Barham,

2013). Many software projects also include a third step: a release process that only allows code that has been reviewed thoroughly enough to be redistributed to the public (Fitzpatrick & Collins-Sussman, 2012).

The second problem is how to make sure that a contributor doesn't have malicious intent in developing the source code. This is also dealt with an effective quality control process. With regard to security, Silic and Back (2013) states, "still, I know for mature projects there is [a] very clear code release process... I guess it would be difficult to change something."

Even though an effective quality control process can help minimize the effects of incompetent or malicious contributions to an open source project, they are not perfect. Silic and Back (2013) also states, "there are millions of lines of code... it would take you an eternity to check it," which indicates that the scale of most open source projects makes it impossible to guarantee with one hundred percent certainty that the code is right. These quality concerns become important whenever software is used in critical applications such as medical devices or critical infrastructure; EEROS's focus on safety provides opportunity for hackers to exploit its features.

## **2.2 Creating and Managing Open Source Software**

In this section, we discuss specifics of an open source project: the tools involved in managing an open source community, and how to protect a piece of open source software. We first explain some of the common tools used by software developers that were adopted by the open source movement, and then move onto software licenses and how they offer protection for both the original owner of the software, and contributors to the same piece of software. We do this to afford the reader the opportunity to understand how EEROS will function as an open source project.

### **2.2.1 Tools and Community Involvement**

We have discussed the ideologies and concerns of the open source movement, and the common elements between different open source software projects. However, we've neglected to mention how open source software projects (and software projects in general) use technology to their advantage, especially how EEROS can use existing technology to help community members communicate.

The key element of any open source project is communication. When dealing with hundreds of contributors who come and go as they please, with no predetermined task or directive, communication between contributors is very important. But conventional communication methods don't work for managing such a large community, especially when the community is geographically diverse. Imagine how much time someone would spend on the phone trying to coordinate everything between different groups of the community. Face to face meetings would suffer the same fate as phone calls: the amount of time needed for these meetings would eat up the entire productivity of the community. Instead, open source projects

rely on forms of asynchronous communication, such as E-mail, mailing lists and newsgroups, or Internet forums (Bonaccorsi & Rossi, 2003).

The scale and chaos that make communication between members of an open source project difficult also makes collaboration between the same members tricky as well. If all the members of the project simply worked on different parts of the project in parallel, and agreed on how everything should fit together, then there wouldn't be any problems trying to assemble a final project. But this relies on writing down exactly how hundreds of different parts of the project are going to fit together before knowing exactly what each part does. This is nearly impossible to accomplish. Instead, open source projects rely on two different ideas: object-oriented programming (OOP) and version control.

Object-oriented programming is the idea that every piece of code that a programmer writes defines how other code can interact with it, rather than just fulfilling a certain objective. By using OOP, members of an open source project can make it easier for other members to come in and understand exactly how to use that part of the code. This leads to a much more modular design, and helps promote coordination between different members of the project (Bonaccorsi & Rossi, 2003).

The other technology that open source projects frequently use is version control. Imagine you have five different people working on editing a paper. They go off on their own and perform their edits and come back to put them all together. Naturally, some of the editors will add material in the same spots that other editors deleted material from. How do you end up actually putting all the edits together? Once you put the edits together, what if you want to go back to the original version? Or maybe you want to go back and only look at the paper if one editor had put in all their edits? These problems are manageable if you only had one round of edits, but what if you had hundreds or thousands of rounds of edits? It would be almost impossible to keep track of all the necessary information. Many people still try and do this in a rudimentary fashion by adding version numbers to their different papers. This is what version control technology does, except on a much larger scale. Many open source software developers use version control technologies like Git and Mercurial (Fitzpatrick & Collins-Sussman, 2012). Git and Mercurial keep track of every single change that occurs to the code of the project, and allow project members to control how different versions of the same file get put together in the end.

These three technologies: asynchronous communication, object-oriented programming, and version control, are hallmarks of an open source project. They provide the means for managing a decentralized, flexible project, which is exactly what an open source project is.

There is another type of technology worth mentioning; open source movements tend to have a centralized website or application that brings together the technologies we just discussed. Websites such as GitHub or BitBucket allow users to upload code via version control, view other code uploaded by users, make comments and have discussions over particular changes to code.

These websites end up being the epicenter of the community; and is an important tool for EEROS to exploit to create a healthy community.

### 2.2.2 Licenses

Another important part of open source software is licensing. In the next few sections, we'll explore what exactly a software license is, why you can't just distribute code without a license and consider it open source, some of the ideology behind licenses, and examine several different open source licenses.

A software license is a statement that declares how the software and the source code can be used, distributed and modified. There are two main types, copyleft and permissive. A copyleft is a license that is designed to guarantee the rights of the user (Phillips, 2009). A permissive license, on the other hand, details what the user can and can't do with that piece of software. There is a significant difference between the two different types of licenses that we'll explore in depth in the next section. But first, why even bother with a license at all?

A license seems unintuitive for open source software. After all, the goal of open source software is to give the user freedom to do what they want with the source code. But without any license, users (and even the original owners) open themselves to litigation (FSF, 2014). If you distribute a piece of software you create, without the license, then someone else can take your original code, copyright it, and then sue you for copyright infringement if you continue redistributing or modifying your own code. This obviously is a barrier to the development of open source software, and is the motivation behind copyleft licenses.

### 2.2.3 Ethics of Licenses

Now we understand why licenses are important in software development. But what's the real difference between a permissive license and a copyleft license, and is one more ethical than the other?

As mentioned before, a permissive license is a license that states what a user can do with a piece of the software or source code. A good example of this is the software agreement you have to read through to use iTunes, or other popular commercial software. If you take the time to read through the entire agreement, you'll notice that Apple, the distributor, protects itself from blame if the software is used for something illegal, if it ends up damaging the user's computer, or if any other similar situations occur. These licenses also typically make sure that the user doesn't try and distribute copies of the software (Holthorf & Kelly, 2009)

The idea of a copyleft license is slightly different. The goal of a copyleft license is to protect the user's intrinsic rights in regards to the software, much like a bill of rights for the user. This is expressed in Stallman's quote: "Think of free as in speech" (Stallman, 2009). These "inalienable" rights of a user are to view, modify, and redistribute the software as the user pleases, provided it doesn't violate any of the same rights of another end user.

Copyleft licenses are more important in an open source setting than permissive licenses. They help form the community behind a software project, rather than protecting the original

“owner” of the project from users who use the owner’s software the wrong way. These copyleft licenses let contributors to an open source project participate in the project without fear of retaliation from the owner.

#### **2.2.4 Examples of Licenses**

In the previous sections, we discussed how copyleft licenses work, and why they are important to open source projects. But how do you write a copyleft license? What goes into a software license in general?

The easiest way to write a license, and understand what a license looks like is to examine several different examples of current licenses. The first of these is the GPL, or GNU Public License (GNU stands for GNU is not Unix). The GPL was developed by Richard Stallman and the Free Software Foundation (Carillo & Okoli, 2008). The latest version of the GPL (version 3) can be found on the FSF website ([www.fsf.org](http://www.fsf.org)). The GPL allows the user to view or obtain the source code, to modify the source code, and to redistribute the source code. It has clauses for handling a mix of copyrighted and GPL software, and requires that any redistributed software is distributed either under the GPL, or more permissive licenses (FSF, 2014).

Another important license is the Apache Public License (APL), specifically version 2. EEROS is written under the APL, so it seems like a good idea to examine its terms as well. The APL is very similar to the GPL, only that it doesn’t give the contributor access to the trademarks of the original source code, so that redistributed software can’t use the same trademarked logos, etc (FSF, 2014).

These are only two of many open source licenses. A comparison of many open source licenses can be found on the FSF website. The full text of the GPL and APL can also be found there.

### **2.3 What makes an Open Source Software Project Successful?**

Open source software is generally provided free of charge, which would seemingly make it difficult to run a successful business around an open source product. In this section we describe how revenue can be generated from the creation and distribution of open source software. First, we discuss a well-known, highly successful open source software company. Then, we discuss some open source revenue models and highlight some companies that have been successful with these models.

#### **2.3.1 Case Study: Red Hat**

Red Hat, Inc. is one of the most well-known providers of commercial open source software. They develop numerous enterprise products such as Red Hat Enterprise Linux (RHEL), Red Hat Enterprise Virtualization, and more recently, Red Hat Enterprise Linux OpenStack Platform. Figure 1 summarizes some of Red Hat’s key milestones since its founding in 1993.

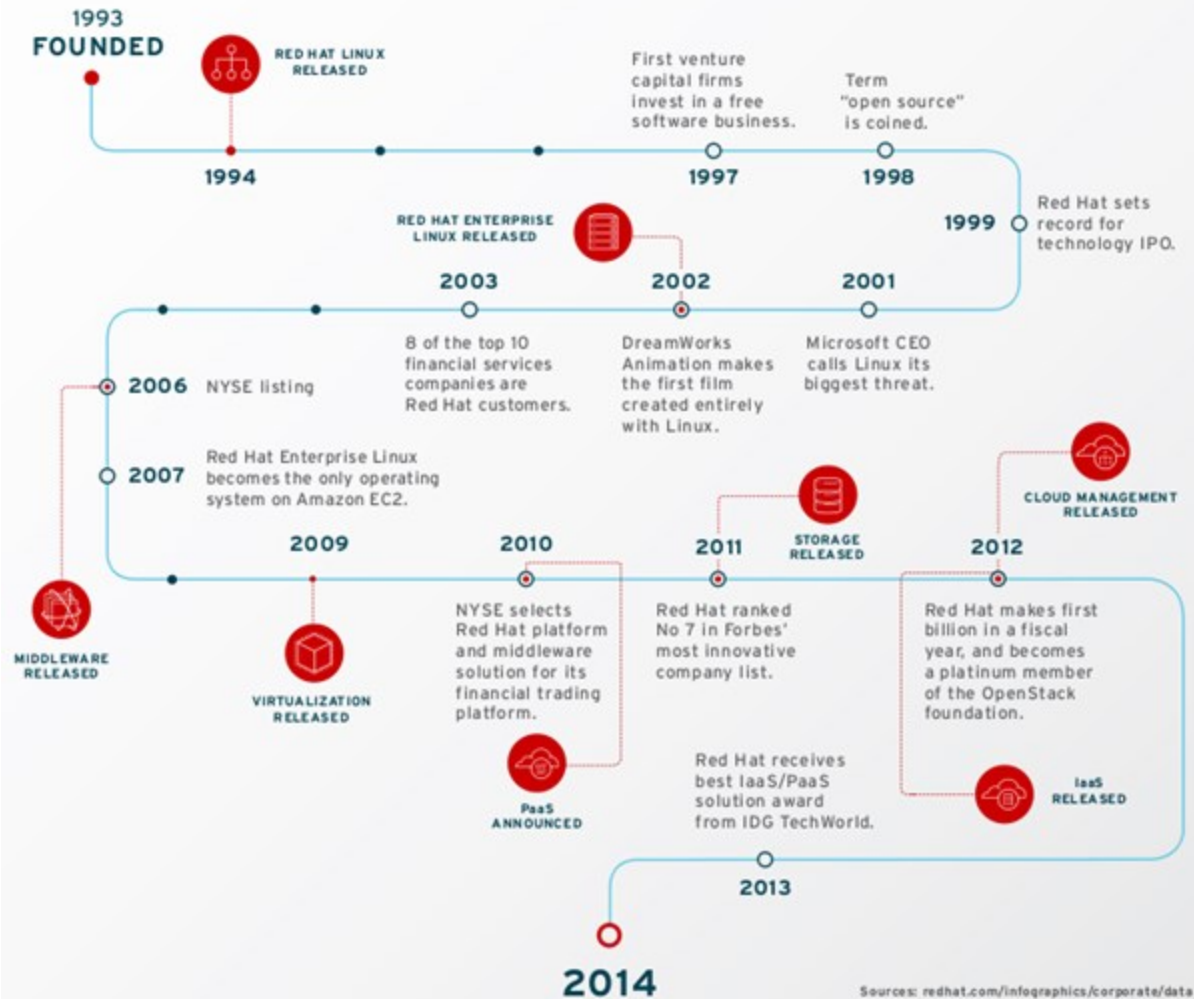


Figure 1: A brief history of Red Hat

By nature, open source software is free of cost, so Red Hat does not make any money from selling software. To a more traditional proprietary software company, giving away their primary product for free seems like a recipe for failure. However, Red Hat's business model depends upon providing a free software product, and selling support subscriptions and training for that product. This model is known as Value-Added Service, which generally refers to a software company that provides an open source platform free of charge, but charges customers for support and other services (Fitzgerald, 2006). Specifically, Red Hat follows the "Support selling" model shown in Table 1. In 2014 alone, Red Hat earned over \$1.5 billion in revenue from these services (Red Hat Inc, 2014). Despite having a different source of revenue than proprietary software, making money in open source depends on the same tenets: building a brand that emphasizes quality of the software and excellence in customer service (Young, 1999).

Since Red Hat provides an open source product, they do not license intellectual property for their own use. However, in a world where software companies are constantly patenting

software, Red Hat must do more than just release their software under an open source license. In an act of self-defense against competitors to open source, Red Hat has reluctantly filed numerous software patents. Along with these patents, Red Hat released a promise not to pursue legal action against those using patented Red Hat software as part of an open source product (Red Hat Inc, 2014).

Red Hat is in a unique position because they are not the only company working to improve their product. Due to the open nature of Linux, other large companies often have it in their interest to make improvements to the Linux system. By making these improvements for themselves, Red Hat can adopt these improvements and integrate them into their own system. Jim Zemlin, Executive Director of the Management Team for the Linux Foundation, puts it nicely in an article featured on Linux.com:

Since Linux has grown, so have the benefits Red Hat receives (and gives to others). When Facebook contributes code to make their data centers more efficient, Red Hat benefits; when Red Hat contributes code to improve file systems, mobile device makers benefit; when mobile device makers contribute code to improve power consumption, super computer cooling costs go down; when super computer users contribute code to make Linux faster, Wall St. benefits with faster trading systems -- and so on and so forth. So you can see that the positive feedback loop that is represented in the billions of figures above shows no signs of slowing down. (Zemlin, 2012)

Every year, the Linux Foundation releases a report called “Linux Kernel Development: How Fast is it Going, Who is Doing It, What They are Doing, and Who is Sponsoring It.” In 2013, it was found that there were over 10,000 unique developers that contributed to Linux, and they were sponsored by more than 1,000 different companies. According to this report, Red Hat was the number one company sponsor since the last report in 2012, where Red Hat was also number one. In fact, Red Hat has been the number one company contributing to Linux since the Linux Foundation began releasing this report in 2008 (Corbet, 2013). Red Hat uses this community to drive their business model; with a strong community, EEROS could derive success as well.

### **2.3.2 Other Open Source Business Models**

Red Hat holds certain lessons for EEROS, but Red Hat is not the only successful company providing free and open source software. Other companies, like Google and Canonical (the developers of Ubuntu Linux), also make money while distributing open source software, but they do not follow the same revenue model as Red Hat. Most open source software companies will follow one or more of the eight revenue models summarized in Table 1. Red Hat mostly follows the support selling model, but they also use concepts from the service enabler model with their OpenStack platform.

Table 1: A summary of open source revenue models (Source: Modified from Rajala, 2007)

Revenue Model	Description	License Types	Revenue Sources
Support selling	Company provides software for free (gratis) but sells support contracts to users.	Any	Media distribution, branding, training, consulting, post-sales support
Loss-leader	Open source software provided for free in hopes that it will stimulate demand for a paid product from the company.	Varies	Other software products
Widget-frosting	Companies that sell hardware provide open source software such as drivers or interface code.	Any	Hardware
Accessorizing	Companies that distribute materials like books or hardware that are associated with open source software.	Any	Supplementary offerings
Service enabler	Open source software is developed and used as a platform to selling consulting contracts or other online services.	Any	Service fees
Brand licensing	Company charges others for the right to use its brand and trademarks in derivative products	Strong reciprocity	Royalties received for brand licensing
Sell it, Free it	Software starts out as commercial, but is eventually released as open source.	Alteration of license type	Initial revenue from commercial product, other models used once converted to open source
Software franchising	Combination of brand licensing and support sellers	Strong reciprocity	Franchiser supplies companies with training and licensing for a fee

Of the eight models listed here, four are important to us because they apply directly to companies that provide both hardware and software, such as those in the robotics and embedded systems industries. The widget-frosting, accessorizing, service enabler and loss-leader models are directly applicable to hardware companies. For example, Arduino makes use of both the widget-frosting and accessorizing models. They sell microcontroller development boards, but provide open source software for working with these boards. In fact, the hardware itself is also open source, but Arduino generates revenue by selling prefabricated boards. They also sell accessories in the form of expansion boards with additional hardware (Arduino, 2015). These

models easily fit the EEROS ecosystem; revenue could be generated by selling robots and additional software, while the EEROS software platform is distributed for free.

### **2.3.3 Crowdfunding and Open Source Software**

In recent years, many new products have been successfully funded using crowdfunding. Crowdfunding calls for many individuals to make small financial contributions to a project, and in return, individuals will generally be able to get the final product at a reduced price or receive some other kind of compensation (Mollick, 2014). Often these projects look to crowdfunding websites, like Kickstarter or Indiegogo, to simplify the crowdfunding process. These websites allow project founders to create an advertisement for their product and set up different contribution levels with rewards based on the amount contributed. For example, OUYA, an Android-based video game console, was successfully funded on Kickstarter in 2012. The project founders offered ten different reward levels, ranging from a \$10 to \$10,000. The rewards for these contributions ranged from being able to get a username before launch, to receiving one of the first production models, to getting the contributors name engraved on all models from the first production run (OUYA, 2012).

Although it depends on some open source software, OUYA is not primarily an open source product. One open source project that was successfully funded is Parallella, an open source hardware and software product developed by Adapteva that aims to make research in the field of parallel computing easy and affordable. It was successfully funded on Kickstarter in late 2012, and like OUYA, it offered a wide range of contribution levels. The important thing to note is that Parallella follows the widget-frosting and accessorizing revenue models described in the previous section (Adapteva, 2012). The accessorizing revenue model is often used in crowdfunding situations, and the widget-frosting model is common among hardware products, especially robotics.

These examples show that crowdfunding can be an effective way for a new product to gain traction and popularity quickly. If a product has a devoted community, a crowdfunding campaign can successfully fund the initial development and manufacturing costs for a new product.

## **2.4 The Robotics Community**

The world of robotics is in many ways similar to the world of open source software, described in section 2.1. Both seem to have developed a sense of community between innovative minds that want to contribute to the world around them. First, we discuss the history of the robot itself, and how that history has managed to create such a community. Then we move on to discussing aspects of the community itself and the kinds of applications that robots see in industry and education today.

### 2.4.1 The History of the Robot

The development of the robot has greatly impacted the industrial world since its introduction in the 1950's. Historically, the robot is not really a novel development, but rather a natural development of human engineering. When the production line was developed, it left workers toiling with monotonous and often dangerous tasks for long shifts. Robotics began with automation, as manufacturers tried to use conveyor belts and other mechanical systems to try to accomplish tasks with machines instead of humans. In the late twentieth century, advancements in computing made artificial intelligence possible, and the robot as we know it soon followed. Since then robotics has had a profound impact on the world, changing everything from the industrial world to our everyday lives.

In 1495, Leonardo da Vinci designed what is accepted to be the first humanoid robot, a machine made in man's image (Sargent, 2015). However, the actual word 'robot' wasn't used until 1921, when Czech dramatist Karel Capek created a play in which mechanical beings created by humans are used to serve their makers (Sargent, 2015). The first development in modern robotics was seen in 1946, when John von Neumann developed the concept of stored code, i.e. a computer program. This would allow a robot to repeat a task indefinitely, using just one program (Sargent, 2015). At this point, mechanical and electrical engineering had advanced enough to support complex systems. The development of computer science had been the missing link, and as it advanced, robots became ever more possible.

With the groundwork for robotics established, it was of no real surprise that the first industrial robot was in operation by 1961. Called Unimate, it was essentially a gigantic arm used by General Motors to move and weld die-castings on an assembly line, a dangerous job for humans. Unimation "designed and machined practically every part in the first Unimates. They also invented a variety of new technologies, including a unique rotating drum memory system with data parity controls" (Unimate, 2003). From here the world of industrial robotics was born. It would develop over the next half century to become a highly advanced field, with applications in nearly every manufacturing or other industrial field.

Today, industrial robotics has had a massive impact on the manufacturing world. Nearly any assembly line can be seen using robots in some way to assist product creation. Like their Unimate predecessor, most industrial robots are used to do jobs considered too dirty, dull or dangerous for humans (Spencer, 2002). Robots can also accomplish tasks at a level of precision that humans cannot match; an example of this is the automotive industry. Parts need to be welded and fitted together with extreme precision, as well as pieces painted and fastened. These tasks are not only dangerous, but also tedious (imagine a worker fastening the same screw on a car model for nine hours every day) (Spencer, 2002). While most industrial robots are nothing more than pre-programmed robotic arms, there are emerging examples of robots that are more autonomous that can act as a worker on a factory floor. This continually developing world of industrial robotics has indeed given rise to a robotics community, who share a passion for robotics, industrial or not, and artificial intelligence.

### **2.4.2 Social Aspects of the Robotics Community**

When the history of robotics is analyzed, it is no surprise that a thriving community has developed around robots. Recall that the limiting factor on robots has typically been the limitations of computing and software. This means that researchers working on robots were typically software-oriented and had close ties with open source community members. As the open source movement developed, the ideologies shared by open source enthusiasts were also prevalent in those researching with robotics. This association can be seen in the robotics community today and in the culture surrounding robots and technology.

The robotics community has been influenced heavily by a recent cultural shift. As industrial robotics grew over the years, the sci-fi movies of the 50's seemed closer to reality than fiction. People's interest turned to awe as the abilities of machines surpassed that of man. In the past fifteen years or so, this has given rise to a new phenomenon: the rise of "Geek Culture." Sharply contrasting the mainstream idea of being "too cool for school," it has become cool to be a geek, and this has led to people sharing and celebrating things like robotics; something not accepted under the previous definition of cool (Harrison, 2013). This ties in closely with the open source software movement; geek culture is run by younger people interested in STEM fields, and they are naturally drawn to the cutting edge areas. In the past ten years, robotics and software have seen huge developments, so it is no surprise to find that these Open Source advocates overlap with robotics communities on the web.

There are several great examples of online communities who share amongst themselves the recent developments in robotics as well as their own homemade robotic creations. Community of Robots, RoboCommunity, and the Robotics SubReddit all feature some sort of open forum where members contribute stories, articles, pictures and other media. The emergence of these communities closely ties in with the Open Source movement, and they share the same ideologies. Magazines such as Make claim to support the 'Makers Movement', an analog to software's open source movement. A good example of this is shown in Make Magazine's fourth issue, sporting the phrase "If you can't open it, you don't own it" as the title of their so called 'Maker's Bill of Rights' (Jalopy, 2005). This phrase elegantly describes one of the ideologies shared by the robotics and open source communities.. They do not want to be just consumers; they are capable of innovation themselves, and they want to contribute to the world of technology with their own thoughts, ideas, and inventions. According to a member of these communities, if you simply use someone else's proprietary software or buy someone else's machine, it's not really yours (Jalopy, 2005). You can only claim ownership of something you helped to create, and these communities take pride in being able to produce their own innovations.

### **2.4.3 Differences Between Education and Industry**

With the ideologies and social aspects of the robotics community in mind, it is time to discuss in greater detail where robots are used. Even in recent years, robotics has seen nothing

but growth. In 2007, 6.5 million robots were in operation worldwide, with a predicted 18 million in 2011 (Guhl, 2009). This growth comes from a symbiotic relationship between two main areas of robotic use: education and industry. However, these areas are near polar opposites of each other in terms of goals and applications. Industrial robots represent the pinnacle of safe, reliable, robust, well-tested and precise machines. They are built to perform a single task perfectly. In the world of industry, there is no room for mistakes, and safety is held in high regard. In addition, accountability is a major issue in industry. If something goes wrong, it needs to be absolutely clear what caused the error so it will not be repeated in the future. These aspects make designing and programming any industrial robot a challenging process. However, travel to a robotics laboratory in an educational institution, and you will discover a vastly different scene.

Parts and tools are strewn amongst papers, hand-drawn designs and hastily scribbled notes and calculations. A prototype, held together by zip ties and duct tape, sits on a table, ready for testing. This is a sharp contrast to the highly regulated and efficient industrial world. This is where innovation happens: students and professors work together to push their robotic invention to new levels, to accomplish something no one else has done before. Perhaps this robot can communicate in real time with others, or it can navigate a room full of obstacles by using a complex array of sensors coupled with advanced algorithms. Here, a failure is seen not as an error, but something to be learned from. Safety is still important, so it's best to keep your distance in case something does go wrong. It may seem like the educational world is largely inferior to the industrial one, but in this chaotic laboratory the envelope is pushed and innovation is born. Ten years from now, a similar concept, now rigorously tested and refined, may be used in some industrial application. These two worlds need each other to evolve and continue to grow. The educational world produces important research that will allow the next generation of industrial robots to evolve. At the same time, the demand for these new robotic advancements wouldn't be nearly as great without any industrial applications, and projects don't get funded for their cool factor.

As different as these worlds may be, there is certainly one factor that every robot shares, that helps define the very nature of a robot: it uses software. Today, there are many options for robotics software packages, many of which are designed with certain applications in mind. And given what we know about the Open Source and robotics community, it was only a matter of time before someone developed open source software that could be used on almost any robot. This software, called ROS, is one of the most widely used tools in educational and industrial robotics alike, and has to some degree become the standard to which other robotic software is compared. ROS, however, is just one example of an open source robotics framework, community-developed software meant to standardize the way robots are programmed.

## **2.5 Open Source Robotics Frameworks**

Open source robotics frameworks are becoming increasingly common. A simple search reveals that dozens exist, each attempting to solve some set of problems within the robotics

community. Several companies even create robots designed specifically to be used with their frameworks. This section examines some of the largest contenders in open source robotics software: what they have done well, where they have fallen short, what licenses they have used, and other aspects. This helps us to better understand the market EEROS will be entering and what competition it may face.

### 2.5.1 ROS

ROS, the Robotics Operating System, is perhaps the most successful open source robotics framework on the market today. It is a collection of tools, libraries, and conventions which aim to simplify creating complex and robust robotic systems. ROS provides hardware abstraction, low-level device control, message passing between processes, package management, as well as other standard operating system services. Keeping with its open source nature, ROS has been designed mainly to run on Unix-like operating systems such as Ubuntu Linux (ROS, 2015). ROS has four main goals, the first of which is easy integration with other robot software frameworks. This is one of the major reasons why ROS is so successful. Language independence is the second goal, which is accomplished by allowing users to write code in C++, Python, or Lisp. The third goal, easy testing, is realized by the built in test framework *roctest*. Scaling, the final goal, allows for ROS to be used in large systems and development processes (ROS Introduction, 2015). These features and goals make ROS desirable to developers and roboticists.

Development of ROS began in 2007 at the Stanford Artificial Intelligence Laboratory under the name *switchyard*. In 2008, development was moved to Willow Garage where it remained until 2013. While at Willow Garage, ROS saw enormous growth, and Willow Garage has even released several robots designed especially for use with ROS. Their success would not have been possible without the collaboration of researchers at over 20 institutions. In mid-2013, ROS development was moved to the Open Source Robotics Foundation where it currently remains (ROS History, 2015).

ROS has been quite successful since its start in 2007. Their vast list of contributors continues to grow and improve ROS. It has become a popular tool among educators, and is even a core component of several classes at WPI. ROS's flexibility has allowed for integration with other open source project libraries, including Gazebo, OpenCV, Orocos, and others. Aside from these successes, ROS does have several drawbacks. It exists as a top layer program, and it needs third party software like Linux or Unix to run, which introduces undesirable overhead. ROS also lacks real-time support, meaning that it cannot guarantee a response to an input within a strict time constraint. ROS can, however, be integrated with real-time code from other project libraries. Regardless of its drawbacks, ROS has managed to dominate the market for open source robotics frameworks (ROS Wiki, 2015).

### 2.5.2 ROS-Industrial, Urbi, and The Player Project

ROS is a powerful piece of software, but it can't do everything by itself. The following three robotics frameworks can be used to add advanced functionality to ROS, or can be used as standalone pieces of software.

ROS-Industrial is an open source project with the goal of extending the capabilities of ROS to manufacturing. Since January 2012, this framework has been designed to address the problems caused by the limited software architectures of industrial robots. Despite being incredibly flexible and robust machines, the variety of tasks industrial robots are capable of has been restricted by poor investment in software. ROS-Industrial aims to solve this problem by creating a platform for interoperability, meaning that industrial robots will be able to “speak the same language” regardless of their manufacturing origins, equipment, sensors, or desired task. Quantifying the success of this software is difficult, as it has not been in development for very long. ROS-Industrial, as the name implies, relies heavily on ROS, and thus shares many of its good and bad attributes. Like ROS, it provides advanced libraries for free and offers a way for academic research to quickly be applied to industry. It shares the cost saving benefits of open source, but still lacks native support for real-time control (ROS-Industrial, 2015). Overall, ROS-Industrial is a new and promising robotics software framework.

Developed by Gostai beginning in 2003, Urbi is an open source software platform used to create applications for robotics and complex systems. Urbi is fully compatible with hardware supporting Windows, Mac OS X, and Linux, giving the user more flexibility than ROS in choosing an operating system. That being said, code for Urbi can only be written in C++. Support for ROS has been integrated into Urbi, allowing them to share functionality. An interesting aspect of this software, though, is its licensing system. It uses a dual license model which gives the software to individual users for free while organizations must pay Gostai for support and advanced features (Gostai, 2015). What Urbi seems to lack though, is the sense of being driven by community that is so prevalent in ROS and other open-source projects. Development of Urbi appears to be the sole effort of Gostai, as no mention of community development is found on their website (Gostai, 2015). This means that Urbi might be limited in growth potential compared to other open source projects that have a thriving community since community plays such an important part in the development of open source software.

The Player Project is “probably the most widely used robot control interface in the world.” Utilized by over 100 laboratories around the globe, the Player Project aims to create free software for research in robotic systems and has been developed by an international team of robotics researchers since 2000. The project has created three pieces of software: A robot device interface called Player, a multiple robot simulator called Stage, and a 3-D multiple robot simulator called Gazebo. Player is unique in that it allows robot control programs to be written in any language and to be run on any computer with a network connection to the robot. The Player Project has received funding from numerous sources including the National Science Foundation (NSF), the Defense Advanced Research Projects Agency (DARPA), the Office of

Naval Research (ONR), and the Jet Propulsion Laboratory (JPL). It is licensed under the GNU General Public License, making all code free to use, distribute, and modify. Like many other robotics software frameworks, support for ROS has been integrated into the Player Project, allowing ROS users to simulate their robots with Gazebo and Stage (PlayerStage, 2015).

### **2.5.3 The Orocos Project**

Starting in December 2000, The Orocos Project (Open Robot Control Software) has aimed to develop a free, general purpose, modular framework for robot and machine control. The objectives of this project are to be component based, multi-vendor, and to focus on real-time control of robots and machine tools. The real-time control aspect of Orocos makes it unique, and is also why Orocos is a valid competitor of EEROS (E. Nielsen, personal communication, January 30, 2015). Orocos is also able to give ROS real-time capabilities. This software is targeted towards framework builders, components builders, application builders, and end users. Operating under the GNU Standard C++ Library license, Orocos has strived to be “industry friendly”, allowing applications created with Orocos to remain the property of their respective creators (Orocos, 2015).

The libraries of Orocos include the Real-time Toolkit, Components for control, Bayesian Filtering Library, and the Kinematics Dynamics Library (see Figure 2). The Real-time Toolkit is the infrastructure that allows the user to build robotics applications in C++. The Components Library provides pre-existing components including those for control and hardware access. The Bayesian Filtering Library provides application independent framework in Dynamic Bayesian Networks. This is useful for recursive information processing and estimation algorithms. The Kinematics Dynamics library allows the calculation of kinematics in real-time. This enables easier control of robotic arms and manipulators. These four components of Orocos give users the tools they need to easily create complex real-time robotics control applications and have made The Orocos Project the success it is today (Orocos Overview, 2015).

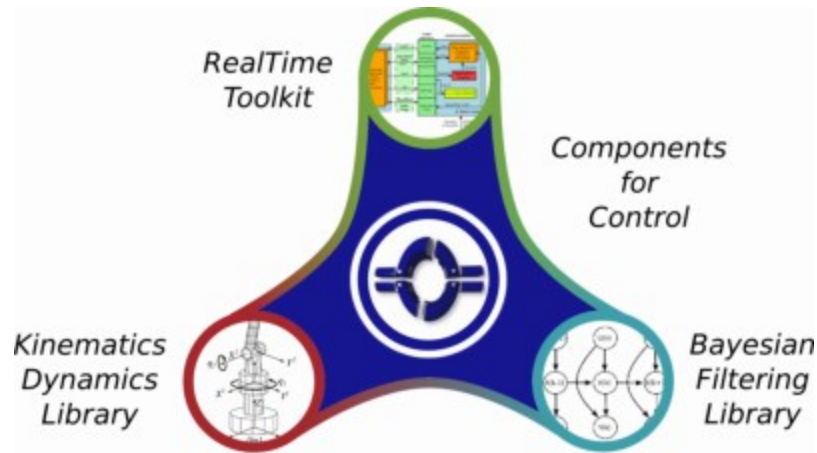


Figure 2: The three main components of Orocos

## 2.6 EEROS

We hope that the reader has gained a broad understanding of the world of software and robotics, and how certain projects have succeeded. Now we address the software at the center of our project: EEROS. Understanding the world in which this software will be used is crucial in understanding what it hopes to accomplish. With this in mind, we try to see how EEROS can fit into the world of robotics.

### 2.6.1 Introduction to EEROS

EEROS, as described on its website, is an open-source software solution for the development of educational and industrial robots (EEROS Team, 2014). This sentence neatly consolidates the goals of the researchers at NTB University of Technology, Buchs & St. Gallen into one simple expression. The first thing mentioned is the open-source nature. The researchers believe that EEROS should not be proprietary; rather it should be fully available to the robotics community. EEROS is intended for both educational and industrial use. This is a broad target audience, meaning that the software must be extremely flexible to deal with the different worlds of industry and education discussed in Section 2.4.3. Indeed, the website goes on to indicate that EEROS is extremely flexible, and that the acronym, EEROS, is itself indicative of its advantages over competitors. It stands for Easy, Elegant, Reliable, Open and Safe (EEROS Team, 2014). These qualities are the foundation of what makes EEROS an exceptional piece of software.

EEROS is foremost designed to be easy to use. It should not be difficult to implement the software into a project. This alone can doom a software project that may have otherwise been successful. Without ease of use, a potential user will become frustrated and move on to another program. The elegance of EEROS is seen when looking at its core design. It integrates three sub-systems: “The Core System, the Safety System and the Sequencer Framework” (EEROS Team, 2014). By having separate systems for core interactions, safety guarantee and

real time support (sequencer) EEROS has a leg up on the competition. This kind of safety system or real time sequencer is usually built on top of a framework, rather than integrated with the core system. EEROS allows for better safety guarantees and more reliable real-time support. This alludes to the third attribute: reliability. If software is unreliable, its chances of success are greatly diminished. If software is reliable (works as intended nearly all the time) it shows legitimacy, which is important for new software. We have already talked about the importance of EEROS being open. Community driven software naturally appeals to the robotics community, and it provides for greater potential impact than a proprietary release. Finally, EEROS must be safe. No industry will give a second thought to an unsafe program, so EEROS needs to be rigorously tested before it is ready for commercial use. However, the integrated safety sub-system allows for EEROS to become safer than its potential competitors. This is one of the key problems that EEROS aims to solve.

### **2.6.2 Problems EEROS Attempts to Solve**

One of the biggest requirements for software to succeed is that it should solve some problem that other software has failed to solve in the past. This is really an extension of simple economic logic; why would a consumer buy your product if another product he or she already has can do the same thing? EEROS does solve some important problems, but without the background discussed previously it may mean little to the reader. Other robotic software is safe, so what makes EEROS special? By nature, EEROS is unlike many other programs and software that has been developed for use with robotics. EEROS is a framework, and acts like an operating system for the robot. Other programs, like one that may move a robotic arm, are executed within the framework. By integrating the safety features directly into the framework, EEROS lets developers define events (like a sensor, or emergency stop button) that would either stop the robot from moving or command a change in its actions (EEROS Team, 2014). In addition, EEROS defines a continuously updating safety level which evaluates based on sensor input how safe the workspace is, and lets a developer define events based on the safety level.

Let's look at a simple example. A robot consists of a large and powerful arm that can lift the frame of an automobile. As a worker walks closer, a proximity sensor's input tells EEROS that the workspace has become less safe. As a result, the speed of the arm slows down, eventually stopping when the area is at a critically unsafe level. This kind of integrated safety feature allows a developer to define events and outcomes that will override the current program (EEROS Team, 2014). By having a safety feature built in at the framework level, EEROS is inherently more safe than its competitors. EEROS has another feature that gives it an advantage over its competitors; it supports real-time robot control. Other systems provide complicated ways to add real-time support, but EEROS goes a step further by building these tools into the base system. Again, an example may help illustrate this problem.

Say an assembly line robot running competitor software receives two inputs called input A and input B. Although these are simultaneous, the line-by-line nature of computing forces the

robot to process them sequentially. This can lead to problems where the output becomes out of sync. Programs that can prevent this are said to offer real time support, and this can be an important feature in an industrial setting where even small errors can be costly. EEROS provides real-time support by using a time domain system (EEROS Team, 2014). By attaching every block (blocks can be inputs or signals) to a time domain, it accounts for these small differences by slowing certain blocks in reaching their destination to assure that the program executes computations correctly in real-time. Figure 3 shows the processing of two input signals in a single time domain.

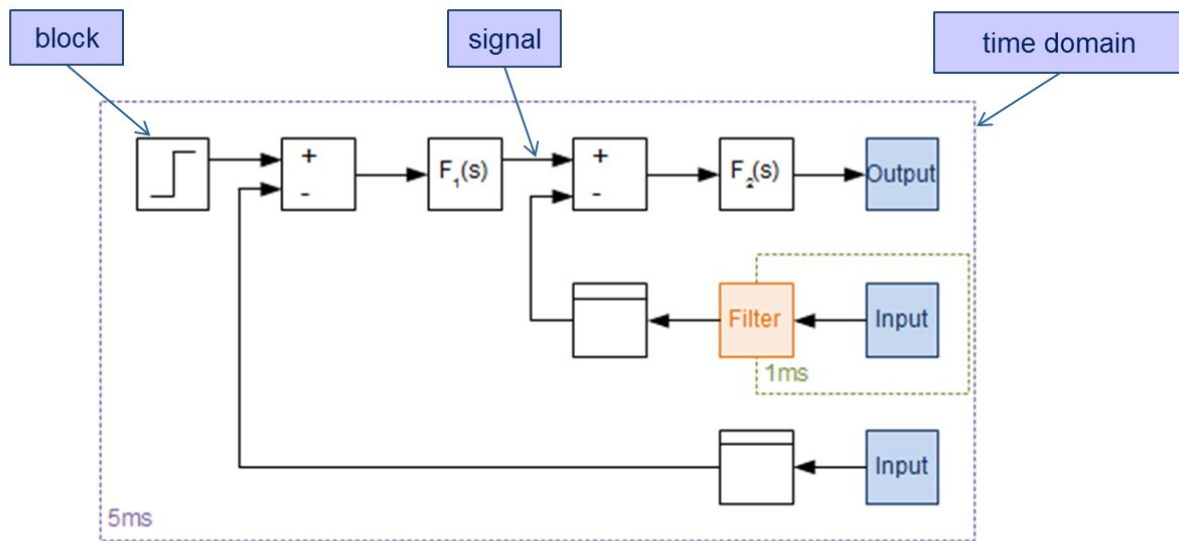


Figure 3: The relationship between blocks, signals, and time domains

The differences between EEROS and its competitors are not easily understood, and to the layman may seem too trivial to make any real impact. However, in the industrial world these differences are enough that EEROS could become a new global standard for robotics control software. By being safer and taking a novel approach to real-time support, EEROS has the potential to be extremely successful.

### 2.6.3 Current State of Development

EEROS is still in the early stages of development. In 2014, selected users were allowed to use the first prototype version of EEROS. Additionally, the EEROS development team has started an open platform called EEDURO, the EEROS Education Robotics Platform, which aims to create a few different robots that showcase the features of EEROS. Currently, the EEDURO platform consists of a delta robot, shown in the figure below. The EEROS team has expressed that they are developing more robots for this education platform. The EEDURO code and

hardware specification are available for free online, although neither are production-ready at the time of writing (EEROS Team, 2014).

Currently, the EEROS developers have a legal team looking into licensing solutions for the EEROS framework. Until they develop a more comprehensive plan, EEROS will be licensed under version 2.0 of the Apache Software License (EEROS Team, 2014), the same license used by Google’s Android Open Source Project. This license allows complete freedom of use, modification and redistribution so long as a few conditions are met:

1. The license owner cannot be held liable for any damages caused by the software.
2. Trademarks cannot be used by licensees.
3. The original copyright must be mentioned.
4. The full text of the license must be available in any redistributed software.
5. When redistributing, significant changes must be stated.
6. If there is an additional “Notice” file, this file must be included with redistributed versions of the software.

For EEROS, this license protects the EEROS brand while still allowing third parties to use the software and source code in any way they choose.

## **2.7 Building a Community for EEROS**

Many open source projects arise when a group of people collaborate to solve a shared problem. Understanding this problem and how the project addresses it is important in understanding how the project will grow, and what it needs to evolve. EEROS has two potential groups of users that will unite around their software, specifically education and industry. Understanding how these groups might benefit from EEROS is key to understanding what we have to do to help expand the impact of the software, and also provide context to understand how the birth of an open source project comes about.

### **2.7.1 EEROS and the Robotics Industry**

The robotics industry has already been discussed in detail. Now, we are prepared to discuss how EEROS will perform in the commercial world. According to their site, EEROS’s developers say that it is designed to be used in commercial robot systems (EEROS Team, 2014). EEROS addresses the fact that safety is a top priority in any industry using robots. The built in safety feature would be a compelling reason for a developer to use EEROS over a competitor. EEROS is also easy to use by design, meaning that developers wouldn’t be frustrated trying to build their programs on top of the EEROS framework. The real-time support further helps EEROS be a relevant competitor in industrial settings, where precision is of the utmost importance and real-time calculations are ever more prevalent as applications become more complex. The open source nature of EEROS would allow the robotics industry to advance at a faster pace as well. Code that was written by one person could be reused for another purpose, because the nature of open source projects not only allows but encourages this kind of

information sharing. However, the open source nature of EEROS could potentially be a downside if it isn't accounted for.

Open source projects, by their very nature, are not always suitable for commercial use. Even ROS, one of the most successful frameworks, has a separate program entirely (ROS Industrial) to satisfy the rigorous regulations in industry. Given that anyone can update the software with their own modifications, how will EEROS guarantee safety and reliability, two of its most important attributes? It is apparent that EEROS will need to use a variation on the traditional open source methods to be successful in industry. A potential solution would be to offer a guaranteed stable version of EEROS, while allowing the community to make changes to a second, more experimental version. Major changes in the experimental version would have to be approved and tested before being added to the stable version. This way, industries would have a safe and approved version to use, while the community would grow the experimental version separately. This method is nothing new; Red Hat develops Fedora Linux for non-enterprise users. This Linux distribution is used as a testing platform by Red Hat to gauge the success of new features before integrating them into their larger Red Hat Enterprise Linux product. A similar system may work well with EEROS.

Industrial platforms need to have accountability as well. With an open source project, it is not always clear who is accountable when something goes wrong. The aforementioned system would help with this, as industries would know that they could contact the producers of EEROS with problems they have using the stable version. There is also the problem that individual companies may not want their edits to EEROS to be available, given the time, energy and funding that they use to develop solutions to their problems. While it doesn't strictly follow the open source doctrine, a compromise must be made between proprietary and open source if EEROS is to succeed in the commercial world. One solution would be a kind of "app store," where companies could charge for programs and algorithms that they wrote for their own applications. Of course, contributors could still update EEROS under the open source license, and could offer their algorithms for free so others could use their work instead of solving the same problem again. This allows EEROS to proceed in an open source manner, but uses a model that has been successful in the commercial world already. With this kind of modified open source outlook, EEROS would be able to fully be applied in the industrial world of robotics.

### **2.7.2 Potential Funding Options in Industry**

With any project comes the question of funding. EEROS is no different. Being able to fund the project is an integral part of its success. As of now, the development of EEROS has been funded by grants (EEROS Team, 2014). However, once it is fully developed and ready for the public, EEROS will need new sources of funding. Many open source projects come up with creative ways to make money. Since they don't charge for the software itself, often they look to make a profit from areas that are often taken for granted in proprietary software. These include

customer support and help features. This method, called the Service Enabler revenue model (as detailed in Table 1) would be an excellent candidate for EEROS. As discussed before, Red Hat uses this method to fund their Linux software, and is quite successful in doing so. Linux and EEROS both act as operating systems, or platforms for applications to run on, so this revenue model could be a promising choice.

We should also look into the Widget-frosting and Accessorizing revenue models, also explained in Table 1. These are both examples of open source models that have seen success in hardware related open source projects. While EEROS is software, it is closely tied to robotics hardware, so these are relevant revenue sources to consider. In a way, the EEROS team has already started using this as a revenue model. The team does contract work for companies that require specialized robots; they build them at NTB and program them with EEROS (E. Nielsen, personal communication, March 16, 2015). Other open source projects simply ask for payment or donations based on the honor system. While more risky, this method has seen success because it trusts the goodwill of the users, most of which are appreciative enough to give back. With a combination of some or all of these methods, we believe that EEROS can be funded successfully without being dependent on grants.

### **2.7.3 EEROS and the World of Educational Robotics**

To have the greatest impact that it can on the robotics community, EEROS will need to be useful for educational purposes as well. Since it is a framework, EEROS is designed to be extremely flexible, and it should be able to adapt to educational use readily. There are countless universities and colleges in the United States with strong robotics programs. Among them is Worcester Polytechnic Institute (WPI), with the first ABET accredited undergraduate program for robotics engineering (Dorsey, 2011). We should be looking to build a network of institutions that use EEROS in their programs, perhaps starting at home (with WPI). This will become the backbone of the open source community that we hope to support EEROS with. As is the case with many open source projects, once a thriving community is established the software will begin to develop and advance without assistance. EEROS is currently being shared by NTB Buchs with a small number of partners; once it is ready for release, we will look for other universities and institutions to become partners with NTB in supporting EEROS.

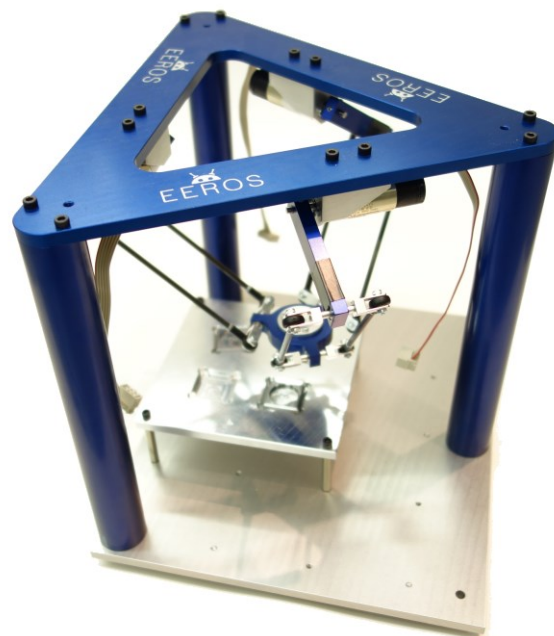
### **2.7.4 Potential Funding Options in Education**

The educational world could provide numerous opportunities for supporting EEROS as well. If EEROS expands to have several universities that use it as an educational tool, these partners could contribute to the project by supplying additional resources and funding. EEROS will need a community that wants to use it and develop code for it. This support, while less tangible than funding, is invaluable to EEROS. Having several institutions that use EEROS would help promote its legitimacy. Furthermore, the institutions that use EEROS will eventually form the backbone of the community we want to build around it. If EEROS is successful, they will contribute many projects, algorithms, and additions due to EEROS's open source nature.

We can also look to obtain financial support directly through the educational world. Every institution that uses EEROS could apply for grants to help fund its development as a promising open source educational tool. This financial and community support make the educational world a worthwhile investment for EEROS's future.

Another avenue worth exploring is the use of advertising for EEROS in the educational world. Holding expositions and similar events should be part of the long term plan for EEROS. This will get the newest generation of robotics engineers interested in using EEROS, and they will bring that interest into whatever company, institution or startup that they join after graduating. This targeted advertising will help secure a long term future for EEROS. This kind of advertising fits well with the open source mindset that we have discussed. The spirit of the advertisement is not to appeal enough so a consumer gets his wallet out, but to involve them in the project and allow them to contribute and learn about EEROS.

The EEROS team has looked into funding prospects in education. They have started to develop a platform of relatively low-cost, open source robots that they call the EEROS Educational Robotics (EEDURO) Platform. They are currently working on two robots, a small delta robot and a seven-axis robotic arm, both shown below. The intention behind these robots is that a school could purchase one or many of them, and use them to teach students how to write robotics software. For example, the EEDURO Delta Robot has a USB port which allows for its users to use either a mouse or XBox controller to control its movements, and it could be possible for students to develop different user input mechanisms. These robots run EEROS; so if students learn EEROS at their university, it is possible that EEROS could become more prevalent in industry once these students graduate.



While many of these things may be farther in the future than the scope of this project, it is important to see the larger picture to understand what this project should aim to accomplish. Our plan has to take steps to ensure the future success of EEROS, and now that we understand what that future success may look like, we are ready to explore more tangible steps to expand the impact of EEROS.

## Chapter 3: Methodology

The goal of this project is to work with the researchers, students, and professors currently developing the EEROS Open Source Robotics Software Framework to expand the impact it will have on the robotics community, and to understand how a developing open source project can transition to a successful project. Expanding the impact of EEROS was a broad goal and it endures past the timespan of our project. Rather than directly expand EEROS, we worked with the EEROS team to identify what they needed to expand the project themselves. From this, we outlined five main objectives:

1. Increase EEROS presence in the robotics community
2. Provide access to sustainable funding
3. Suggest a streamlined and organized development process
4. Analyze the development community health
5. Identify partners for the expansion of EEROS

Our project focused on providing tools that the EEROS team will be able to use to reach their specific goals. Objectives 1 and 2 focused on helping EEROS communicate with the community of EEROS users, while objectives 3 and 4 dealt with giving EEROS the tools it needs to have a symbiotic development culture and meet the needs of the development community. Objective 5 brings these two sides, developers and users, together. Open source software is based blurring the lines between developers and users; it's only natural that both parties are involved in expanding EEROS into community-developed software

Our process for accomplishing these objectives is outlined in the graphic below. Each of our objectives translates into one or more deliverables. In some cases, the deliverables we produced affect more than one of our objectives. The methods required to create these deliverables are outlined in this chapter.

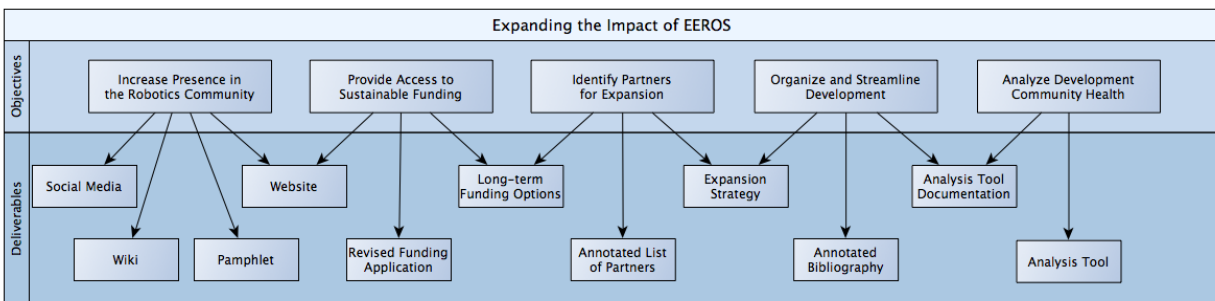


Figure 4: Methodology Outline

Our timeline for the development of these tools is shown below in the Gantt chart. Note that the objectives are presented somewhat chronologically, since the first objectives pave the way for later ones to be more easily completed.

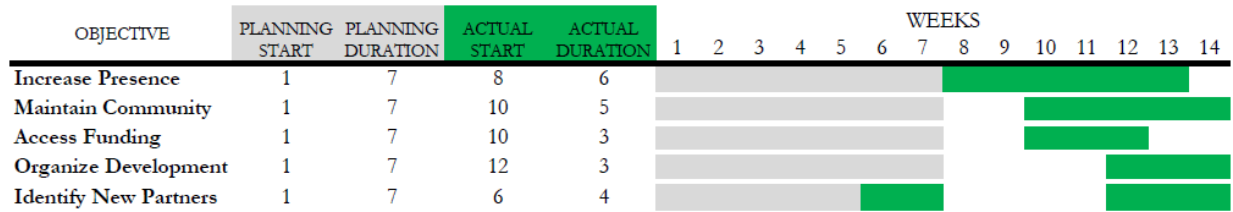


Figure 5: Project Timeline

### 3.1 Increasing EEROS Presence in the Robotics Community

Our background research shows that EEROS is a promising solution to many problems in the robotics community. However, technical promise is not enough to give a software project a significant place in any community. People needed a way to learn about EEROS and become excited about it. We created tools and materials that the EEROS team can use to create a larger presence for EEROS. The tools included:

- A new website for EEROS
- An updated EEROS Wiki
- Social media integration
- A brochure that shows the advantages of EEROS

#### 3.1.1 EEROS Website

Before we arrived in Switzerland, we took the time to interview several WPI Robotics Engineering (RBE) professors, both in person and via email. They were provided with a brief summary of EEROS that we had developed, along with a link to the EEROS website, and asked to answer several questions (included in the appendix). The purpose of the interview was to understand why professors would be likely to use or contribute to EEROS, but we also gained valuable insight into how the professors perceived EEROS based on the website.

What we found was that the professors did not get a good or accurate representation of EEROS from the website. Two of the professors asked for extra information after examining the website; they didn't have enough context to understand why EEROS would be useful. With a properly structured website, these professors would have been less lost, and hopefully more encouraged to use or develop EEROS. The EEROS team was aware that their website was not ideal and a better website could increase EEROS's presence in the open source and robotics communities. Because the EEROS team was busy developing working software, they did not have time to focus on the EEROS web presence. Therefore, our first deliverable to the EEROS team was a new website.

With an improved website design, the EEROS team will be able to use the website as a medium for informing users, developers, and potential investors about the EEROS Real-Time

Robotics Framework. Additionally, we worked specifically towards making the website maintainable, so that the busy EEROS team will be able to keep it updated with minimal effort.

### **3.1.2 EEROS Wiki**

A Wiki is a powerful tool for organizing relevant information related to a topic. A Wiki had already been set up for EEROS, but it wasn't organized effectively to convey necessary information about EEROS. We overhauled the Wiki so that potential users and developers could have a place to find pertinent information. We made three major changes to the Wiki:

1. Reorganize the pages in an easy-to-follow, hierarchical fashion
2. Translate German to English and fix existing translation mistakes
3. Add content where it would be useful

These changes served two purposes. The first was to make the EEROS Wiki more useful to new users or developers. The second was to give the Wiki enough structure for the EEROS team to easily update it by adding new pages or editing existing ones.

### **3.1.3 Social Media**

Social media presence provides a powerful platform to attract new users and developers. Services such as Facebook, LinkedIn, Twitter, and YouTube could provide the EEROS team with an easy way to inform the community about updates and news.

We took the following steps to help EEROS become more recognized within the robotics community:

- Created Twitter and Facebook accounts for EEROS
- Provided ease of management for these services
- Added a blog/news section to the EEROS website

Social media users will see updates on their Twitter, Facebook and LinkedIn news feeds that will guide them to EEROS news in the blog section of the website. This will allow the EEROS team to quickly disseminate interesting and relevant information among the community, keeping current community members informed and attracting new community members.

### **3.1.4 Other Materials**

We found some other miscellaneous avenues for increasing EEROS's presence. One of these was the creation of a Wikipedia page for EEROS. After researching this, we determined that it wasn't feasible at this time. In the future, EEROS may develop enough to warrant a Wikipedia page, so we have set up an account on Wikipedia for the EEROS team to use, and we used the sandbox features to construct some basic features for a future page.

We also created a pamphlet for the EEROS team to use when presenting the project to investors, developers or even users. Having paper materials is a good way to ensure that these potential partners are informed. It also offers a concrete and proven way to distribute information. They can be exceptionally useful at meetings or expositions, so that people interested in EEROS have some materials to remind them about the project later on. We will give the EEROS team the template used so they can continue to update the pamphlet as their project grows and changes.

### **3.2 Providing access to Sustainable Funding**

EEROS is currently funded entirely by grants. To allow EEROS to grow immediately, they will need to apply for more funding in the form of grants. The EEROS team recently applied for large grant called the Horizon 2020, but unfortunately did not receive it. The team was in the process of applying for a grant from the Swiss Commission for Technology and Innovation (CTI). As part of a short-term deliverable, we assisted the EEROS team with the CTI funding application. We primarily proofread for English mistakes, since the German-speaking team has identified English as a potential obstacle. Ensuring proper grammar and phrasing goes a long way in showing legitimacy for a relatively new project. Additionally, we reviewed their development strategy, which has to be stated in the proposal. We found that the development strategy was well thought out, and it aligned with our preliminary ideas.

Looking forward, we provided some suggestions for future funding plans, although these kinds of plans are susceptible to change as the project evolves. As EEROS expands, it will need a long term funding strategy that adapts to its current state. Given the data collected and insights drawn from our background research, we developed a potential funding plan that changes based on certain milestones, such as the first official release of EEROS or the community hitting a certain size. Ideally, there would be three main phases of funding, as shown in Figure 6.

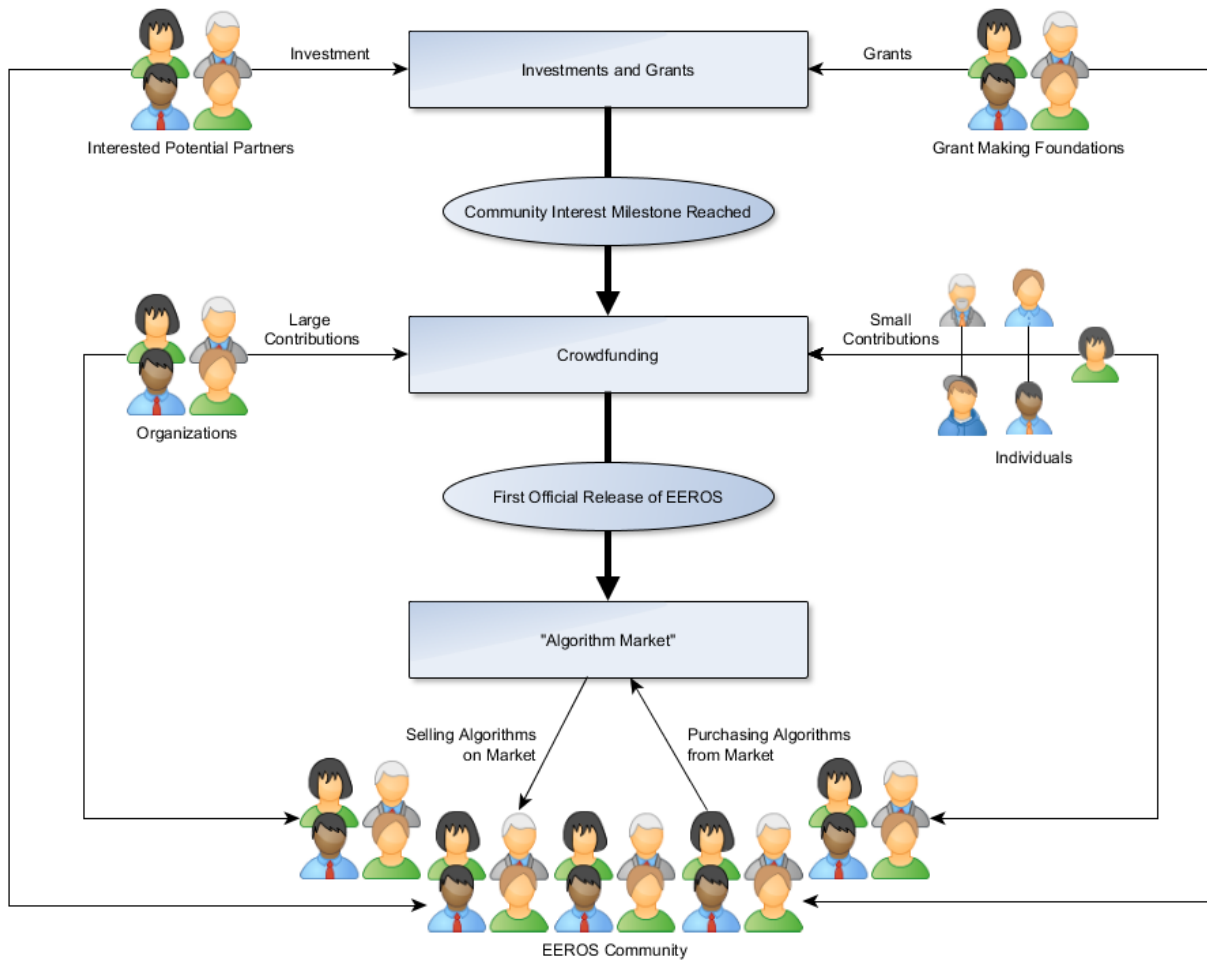


Figure 6: Funding Phases

To ensure access to sustainable funding, we covered both short and long term funding. Originally, we had planned to focus on long term funding, but upon arriving at the project site, we quickly learned that a long term plan is not as useful as the short term deliverable. After all, a long term funding plan is no use if the project cannot acquire funding in the short term.

### 3.3 Organizing and Streamlining EEROS Development

We researched different development methods and related subjects to help the EEROS team understand the different options. To make our findings accessible to the EEROS team, we created a report that summarizes our findings and proposes options for the EEROS team. This document includes:

- Resources on best development practices
- Documentation explaining how to use the community analysis tools

The EEROS team has some development guidelines in place, but as the project evolves, it may turn out that these practices are no longer applicable. For example, when using Git for a small project, committing to the “master” branch is an acceptable practice. However, in a large scale project with more than just a few contributors, it is important to create separate branches of the code and merge them together when features are completed. This helps prevent against breaking the master version of the software. To make adapting to project changes easier, we have created a short annotated bibliography of resources about different development practices, such as the Agile method, Scrum, and Kanban. Scrum is a development model that focuses on short iterations of development called sprints. It also focuses on quick feedback loops and efficient communication between developers. Kanban, which was originally developed for lean manufacturing, creates a pipeline of development where different groups work on tasks from the backlog, passing along their finished iteration to the next stage of the pipeline. Additionally, we have provided sources about open source projects in general. Because the EEROS project is so young, it is hard to predict exactly how it will change. This document will give the EEROS team many different options, so that they can choose which best fits their unique need when the time comes.

The final piece of this report is documentation on how to use the community analysis tools that we have developed. Under the hood, the tool is somewhat complicated, as it uses some fairly complex math and has many software components. However, our hope is that the simple user interface will allow the team to easily view the results from this analysis. We provided a section in the report explaining how to use the tool and how to interpret the various metrics it provides.

This report should help guide the EEROS team through this critical stage of development. By reporting our findings in a formal manner, we made recommendations for the EEROS team based on the work we did in Switzerland. The research we did may be useful for other open source startups as well, because of the broad approach that we have taken. Part of helping EEROS achieve its full potential has been looking at previous case studies as examples, a step that any open source project should take to ensure success. In the future, we hope that EEROS will be successful enough to become an example of excellence in open source.

### **3.4 Maintaining a Healthy Open Source Community**

The EEROS team's eventual goal is to have a strong global community surrounding their project. During our background research we found that a strong community forms around an open source software project that fulfills some common need of that community. Whether or not EEROS will eventually have a strong community surrounding it depends on whether it meets the needs of the robotics community, and whether they are able to communicate how they meet the robotics community's needs successfully (which we addressed with our first objective). However, in all our research, we found that meeting a community's need doesn't guarantee success. The open source mindset typically requires a specific community culture to be

successful, which is very multifaceted. Creating and preserving this community culture involves making sure that the developers of the project are interacting in a positive, productive way. This involves both providing guidelines for developers (which our previous objective suggests) and also making sure that those guidelines encourage developers work in a healthy, productive environment. This is what our methodology is focused on for this objective; being able to monitor whether the development community is interacting correctly.

Monitoring the development community involves keeping track of a lot of different aspects of the community: how they communicate about objectives, how they determine whether code is ready to be released, or how they interact while writing code. Understanding how characteristics of each of these aspects correspond to the health of the development community would be really hard if the EEROS team had no previous examples to look to. Thankfully there are many different open source projects and lots of research on these kinds of topics, see Bonaccorsi & Rossi (2003), Lerner & Tirole (2001), and Athey & Ellison (2014).

Understanding this research is critical for EEROS if they want to be able to judge how well the development team works together, especially as the project gets bigger. However, the EEROS team's primary focus is on developing their software, and they don't have the resources to both develop EEROS and research how best to maintain their community. To serve this need we created a simple tool for them to visualize the state of the development community.

As mentioned before, a typical open source community interacts on many levels while developing code. Unfortunately these interactions can be quite varied between different projects, and creating a tool to measure all of these interactions is well beyond the scope of our work to help the EEROS team. To narrow the scope of this analysis tool, we focused just on how the developers of different open source projects interact while writing code. We developed a mathematical model and a visualization technique to present this kind of information. By having this information, the EEROS team should be able to make informed decisions about which of their development practices are working, and which are causing strife between developers.

### **3.5 Identifying New Partners**

Like any successful open source project, EEROS needs a passionate community of users and contributors. However, in its current state, the EEROS community consists of the core developers, a few clients, and the grant foundation that has funded EEROS so far. Because EEROS is not yet ready for widespread use, we have identified two different types of partners:

1. One or two targeted partners that would be interested in working closely with the EEROS team to either use EEROS or help develop EEROS
2. A database of potential partners that may be interested in EEROS once it is ready for widespread use

The EEROS developers do not yet have the necessary infrastructure in place to support a large community of users and developers. It would be better for EEROS to spread slowly at first to one or two specific partners, since the current state of EEROS would likely require direct communication between the core EEROS developers and any EEROS users.

As EEROS expands, the team will need to reach out to more potential partners to expand their reach in the robotics community. We plan to research different institutions within the different fields of robotics to identify those that may benefit from using EEROS. We created a list of these institutions containing the field of robotics in which they specialize. This list will give the EEROS team options to choose from when looking into new partners, which allows them to create a well-rounded community from the beginning of expansion.

## Chapter 4: Results and Findings

We now look to present the deliverables we created and try to link them to some of the more overarching themes of our paper. Over the course of this project, we worked closely with Professor Nielsen and the rest of the EEROS development team to understand what aspects of an open source community that EEROS could benefit from. We also looked into how being an open source project would fit with their vision of the EEROS software and the kinds of applications it would have. The tools and materials that we have created reflect our efforts to provide the EEROS team a way to establish the culture they want in their open source community surrounding their project. To do this, it was important to keep in mind the culture EEROS is trying to promote: a very diverse community united around trying to provide and use Easy, Elegant, Reliable, Open, and Safe robotics software. This means that we had to make our tools and materials intuitive, i.e. that they represent the ethos of EEROS.

These tools and materials that we provided are grouped by the objectives that we outlined in our methodology. We first present ways in which we worked with the EEROS team to understand how they want to communicate with the community: both in how they want to increase their presence in the community and how they could be supported by the community. We then transition to how we worked with the EEROS team to understand their vision of development: both in discussing important development concepts and providing a way of understanding how the development process is going. We end on information we found about how EEROS can expand to reach new users.

### 4.1 Presence

When discussing our methods for increasing EEROS's presence, we mentioned the existing perception of EEROS from the viewpoint of robotics professors at WPI. The general consensus was that the website was hard to navigate and that the purpose of EEROS was not clear. Our own research showed that information about EEROS was not widely available on the Internet. Without direct communication with Professor Nielsen or another member of the EEROS team, it would have been difficult to use or learn more about the framework. Our solution was to create a web platform as a place for a community to develop around EEROS.

#### 4.1.1 EEROS Website

The EEROS developers had already put together a website containing a wealth of information. However, the site had some organizational and aesthetic problems. To a potential user, the layout made it hard to find relevant information, and the quality of the information was tarnished by grammatical errors from the German to English translation. When we met our sponsor for the first time, he explained to us that the website was created quickly to fulfill a requirement for a funding application, and that a new website was one of our priorities.

After the first week, we had created a working prototype using HTML, CSS, Javascript, and PHP. We used the Bootstrap CSS libraries to create a rich, modern-looking website. The

website was further refined over the following weeks. The new homepage features a short description of EEROS with links to the most useful pages for someone who is new to EEROS.

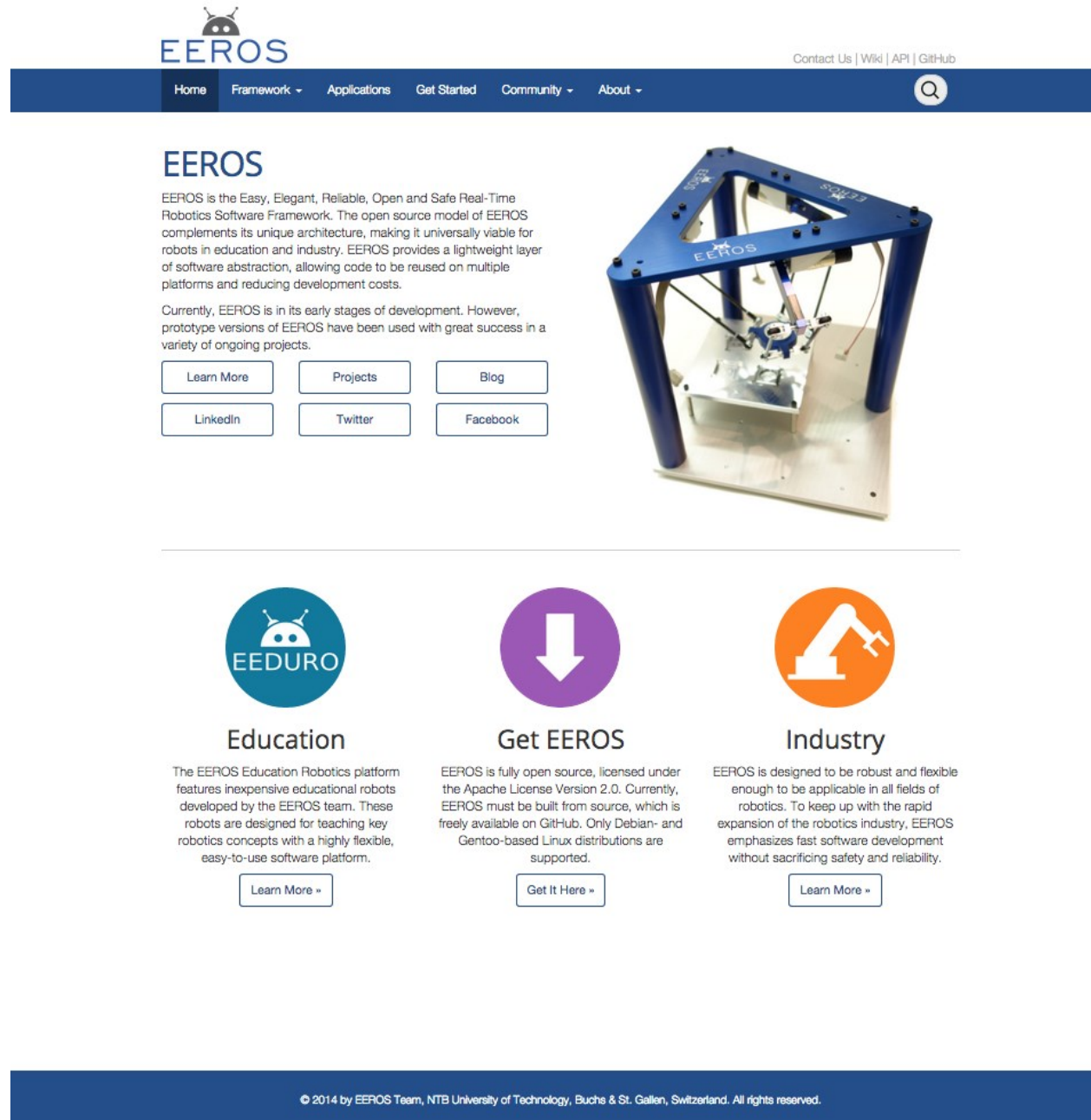


Figure 7: Website Homepage

To see an enlarged version of this image in comparison with the old homepage, see Appendix B.

In addition, we included a page describing the framework, pages on EEROS in education and industry, a page that will help users get started with EEROS, a community section, a section about the EEROS team, and a contact page. Also included are quick links to the Wiki,

Application Programming Interface (API) Reference, and GitHub pages. As previously discussed, the EEROS wiki contains organized information that is useful to both users and developers. The API Reference contains detailed information on the code that EEROS is comprised of, and the GitHub repository is where current and past versions of the EEROS source code are stored. Providing quick access to these resources should greatly improve the overall experience users have while searching for information on EEROS.

To make the website easy to maintain for the EEROS team, we were instructed to migrate the website from being hard-coded to being organized by a Content Management System (CMS). We chose to use WordPress to accomplish this. Aside from making the management of content easier, WordPress allowed us to easily add interactive content, such as forms and a search bar.

Finally, we created a file containing website documentation which will help guide anyone who needs to modify the website. This documentation file includes short tutorials such as “Making a New Page” and “Modifying the Navbar,” along with other useful information and notes. This should serve to ease the learning curve associated with website development and shorten the time required to make edits and additions to the EEROS website. The website is currently online at <http://eeros.org>.

#### 4.1.2 EEROS Wiki

Part of increasing EEROS’s web presence meant having an accessible, useful Wiki for the EEROS community. The structure is now more logical; all information can be easily found from the home page. We also translated pages from German to English, and checked for grammatical issues or phrases that don’t translate to English well. Finally, we added new material in the form of coding guidelines, tutorials, and information about projects that are currently using EEROS.

We started by translating any stray German phrases and consolidating repeated information. The information was present, but it was difficult for users to find what they might be looking for. The EEROS team asked us to address these concerns, so that the Wiki would be production-ready. By making sure that all information was given entirely in English and that it was concise and easy to understand, we helped EEROS further realize its goal of ease-of-use. We continued by organizing these pages in a logical format so that any users could easily find whatever page they needed without navigating the sitemap. We made sure clear links were given to the EEROS website, the API Documentation and the GitHub repository so a user could quickly find these tools and information.

The other major step in revamping the Wiki page was adding content. We added pages about the EEDURO platform and the OmniMoBot to give readers an idea of what projects were currently using EEROS. These pages featured media such as YouTube videos and pictures to help users get an idea of what these projects were. EEROS is applicable to a wide range of robots and robotic solutions, so we want to make sure that potential users see that the EEROS community is diverse and innovative. We also added pages for potential developers; these

included information on coding style guidelines and technical information such as inter-thread data sharing in EEROS. This information was already present in the Subversion (SVN) repository kept by the EEROS team, and it is necessary information for any developer that wants to work with EEROS. We concluded by adding a lengthy tutorial for setting up a DC electric motor to be controlled by EEROS. This tutorial was already made by the team, and by posting it on the Wiki, we have given users an easy way to start using EEROS with a concrete example project. The tutorial shows how to set up many of the important features that EEROS implements, such as the safety system, control system and sequencer. It will provide any potential users with a good starting place for learning how to work with EEROS.

The revised Wiki has had a significant effect on how a user might see EEROS. With this more extensive and well organized documentation, EEROS becomes easier to use, a core tenet of its ethos. Compared to the website, the Wiki is written in a neutral tone, aiming only to inform readers rather than persuade them. This isn't to imply that persuasive writing is inappropriate for the website; rather, that a user will look to the wiki as a trustworthy source since it only acts as a directory. The Wiki will be the tool that developers and users utilize when working with EEROS. It is crucial for open source projects that this kind of documentation is in place, since it helps foster a community around the project. The Wiki will allow users and developers to understand the culture of EEROS and give them the tools they need to advance and use the software.

#### **4.1.3 Social Media**

At the start of our project, EEROS had a small presence on LinkedIn and YouTube. The LinkedIn page was largely unused, and the YouTube channel had many videos but few subscribers. To get the word out about EEROS, a better presence on social media was needed.

We created accounts for EEROS on both Twitter and Facebook. The EEROS Twitter account is @eeros\_framework and the page can be seen at [http://twitter.com/eeros\\_framework](http://twitter.com/eeros_framework). The Facebook page can be found at <https://www.facebook.com/eeros.framework>. To give the EEROS team easy social media integration, we used the WordPress blog functionality with the Jetpack plugin. The Jetpack plugin provides a feature called Publicize, which allows all blog posts to be automatically published to linked social media accounts. This means that when a new blog post is published, posts will be made on social media linking to the new blog post.

#### **4.1.4 Wikipedia Page**

We had planned to create a Wikipedia page for EEROS when we began the project. However, it was determined that EEROS is not yet developed enough to pass the strict guidelines that Wikipedia enforces on its content. All of the currently published information about EEROS comes from NTB, and Wikipedia requires that there be a significant amount of third party sources before a neutral article can be written. We did set up an account for the EEROS developers to use, and we began to format the page in the sandbox feature that comes with the account. Wikipedia should be utilized once EEROS is used and developed outside of NTB.

Wikipedia is a powerful tool that would aid the expansion and recognition of EEROS. Often the best way for software developers to get quick, concise information about a software project is to look at the Wikipedia page before looking at the project's website. Therefore, having a Wikipedia page available could prove invaluable in generating interest from potential partners.

#### **4.1.5 Informational Brochure**

We created an informational brochure for the EEROS team to give to potential partners. Having this kind of material for expositions or meetings can be extremely useful. It concisely highlights the important points about EEROS in a professional manner, and it serves as a reminder for them later on. If they have the pamphlet they are more likely to search our website and Wiki for more information later on.

Informational pamphlets can help demonstrate legitimacy for a new project such as EEROS. Pamphlets or information sheets are commonplace in the business world, and if EEROS is to succeed in an industrial setting its developers should be using this means of communication and advertisement. With this combination of tools, the EEROS team will be able to expand the presence of EEROS readily.

### **4.2 Funding Proposal**

As it becomes more developed, EEROS needs new sources of funding to succeed. In its current stage, these sources are mostly comprised of grants. By working with the EEROS team, we learned how they wanted EEROS to be portrayed. This information enabled us to identify key parts of the Commission for Technology and Innovation Project application that needed to be revised. As stated in the methodology, we revised their most recent funding application by proofreading for proper English and looking for wording that may not translate well from German to English. These kinds of changes can help give EEROS more of an edge in the application process, especially in a country where English is not the primary spoken language.

Since they have not submitted the proposal yet, we do not currently know if they have received the grant. However, we believe that using proper grammar and phrasing goes a long way in making an application more professional and that our efforts should allow whoever reviews the application better understand why EEROS deserves this grant.

Since we have developed tools for measuring the strength and size of the EEROS community, the EEROS development team can use the tools to determine the criteria for the Community Interest Milestone shown in Section 3.2. Our hope is that when this milestone is reached, it will be the ideal time to launch a crowdfunding campaign on Kickstarter or another crowdfunding platform. If there is enough community interest, a crowdfunding campaign should generate enough revenue to push forward to the first official release of EEROS.

If EEROS has a large enough community upon its release, then the "Algorithm Market" should be an effective source of revenue for the EEROS developers. The idea behind the Algorithm Market is to have a platform similar to the Apple App Store or the Google Play Store. Users would be able to submit algorithms that will either be distributed for free or for a price.

Algorithms that are sold would provide revenue for the seller, and EEROS would get a percentage of each sale. In addition, more experimental algorithms, such as those developed within educational communities, would be released freely for testing purposes. This would create a positive feedback loop within the community where industry would be able to purchase algorithms at a lower cost than rewriting them, giving them time to perhaps improve upon some more experimental algorithms. A connection between education and industry like this would open a path for faster innovation within the EEROS community.

In the future, our long term plan will give the developers options for possible funding sources as EEROS continues to expand and become truly community-developed. It is likely that this plan will change, meaning that instead of an absolute plan of action it will instead be used as a guideline. Our sponsor has indicated that long term plans are generally limited in how useful they can be, so we made sure that we emphasized the short term proposal over the long term funding plan. This mindset may prove more useful in the long run, as it gives the EEROS team options so they can adapt to overcome future obstacles.

### 4.3 Organizing and Streamlining EEROS Development

Through our interviews with some of the EEROS users and developers, we learned that while EEROS development has guidelines and standards, many of these are either not followed or too underdeveloped to follow. For example, the EEROS team has stated that they want to use an automated testing system, a common practice for large software projects. However, there are no defined guidelines for testing, so the team rarely writes tests.

Additionally, the team has told us that they want to use an Agile development model. Agile, a movement which officially started in 2001, is defined as any software development model that follows the Agile Manifesto:

We [the authors of the Agile Manifesto] are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more. (Beck et al., 2001)

Because Agile on its own is not a development model, the EEROS team will have to further define their development model. There are many different Agile models, making it difficult to choose the best one for a particular team.

We have developed a short annotated bibliography of resources on topics related to the project management side of software development. We have included resources on coding guidelines, testing guidelines, and Agile development models. Because of the nature of Agile development, it is likely that the EEROS team will need to adapt to changes in the project.

Therefore, providing the team with resources they can use is more useful than creating an entire development plan for them. This document has been provided in an electronic format that the EEROS team can edit as they see fit, and hopefully they can use it as the software becomes community-developed. An adapted version of this document can be seen in Appendix E.

We have discovered that this point in EEROS’s development marks an important turning point. Up until this point, EEROS was small enough that its developers didn’t need to worry about using a specific model. It was already a small community, and as such it was very informal and it used implicit rules for development standards. For the community to expand, the EEROS developers need to make sure they define explicit standards for development and adhere to them. This more formal and explicit definition of practices and guidelines will allow a community to form around EEROS by making communication and collaboration much easier.

#### 4.4 Maintaining a Healthy Open Source Community

Over the course of the term, we developed a tool to visualize the EEROS community. This visualization tool contains data and statistics about the EEROS community based on their own code development. Additionally, the same data and statistics are shown for other open source projects, for comparison. A screenshot of the visualization tool is shown below in the figure below.

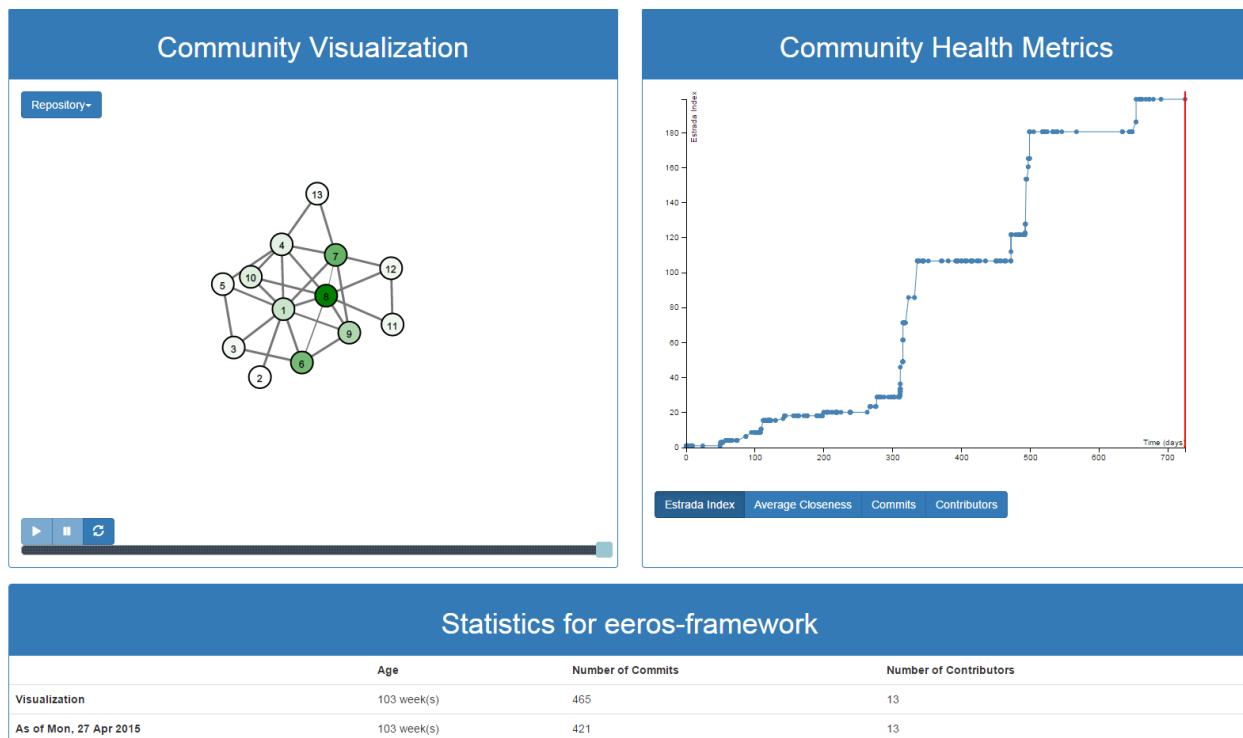


Figure 8: Visualization Tool

The statistics shown are metrics of community health. For example, the Estrada index measures how responsive the community is to contributions. If one person makes a change, do others react, making changes of their own or implementing new things? This represents health from a community activity perspective. A more healthy community tends to have a higher index since they interact more and respond to changes. The other metric, average closeness, is more descriptive as it shows how quickly information can travel between different developers. This doesn't necessarily mean that a community is healthy, just that they are closely knit. In some cases, communities will have several groups of closely knit members who interact very strongly. While individual clusters of developers can be further apart, the community overall can still be healthy, especially in a larger project.

Many of these metrics are derived from the interactions between users. To capture the interactions between users and get useful, intuitive data from them we created a mathematical “network” or graph to model the EEROS development community. A graph is a collection of nodes and their connections (edges). The nodes are the developers of EEROS; edges occur between two developers once they've interacted enough with each other. To create this graph we analyzed the different versions of the EEROS software on the EEROS Git repository. As developers worked on the same areas of the code, we increased or decreased the “value” of their interaction. The interaction value increased when two developers collaborated together, and the interaction value decreased when they contradict each other's work. This allowed us to identify when users are “poisonous” to the project, i.e. when a developer decides to try and implement their ideas when the rest of the community opposes them. We used these interactions to both determine if an edge existed between two nodes and to assign an importance to each developer. This importance takes into account all the interactions the developer has with the community, and how frequently the developer contributes to the project. The specific details on how we calculated these values can be found in Appendix D.

Using these metrics, this tool is designed to provide insight to the EEROS team on how their community fits in with the global open source community. Therefore, we applied our analysis tool to two other open source projects: WPI-Suite and ROS. However, this is a very small sampling of the open source community, so we released the analysis tool as an open source project under a custom license with the intent that it will grow and adapt to the needs of any open source project. As this analysis tool grows, it will allow the EEROS team and other open source projects that use it to understand whether the developers are working together well. As a disclaimer, our analysis tool does not encompass the complete notion of health for a community. There are many other important and relevant ways to measure how healthy a community is. What this tool provides is a mathematical comparison of how developers interact between different communities. While certainly not complete, it is an objective measure of certain important concepts to a project.

## 4.5 Identifying New Partners

The next major step in EEROS's future is to expand to other partners. This is something that the developers have known for quite some time. They have been planning for this move, and through personal contacts and NTB connections, they have a substantial list of partners that may be interested in using EEROS. Several have already expressed interest in using the software as soon as it is ready for release. Part of our work was to catalog these partners into a comprehensive annotated list of partners. This simply lets the team organize their connections and plan the expansion with ease.

Bringing EEROS to North American universities will also help EEROS grow, as many top robotics programs are located at United States universities. We have contacts at several of these universities, so we gave these to the EEROS developers so they can expand to these places. WPI can also serve as an invaluable partner moving forward. We have contacted the Computer Science and Robotics faculty and student. Several professors have expressed interest in using EEROS in their research and plan to encourage students to use EEROS for projects, such as the MQP. The MQP program at WPI is a great platform for further developing EEROS. This would provide dedicated users who could also help develop the software and make suggestions for future improvements. This kind of project would be a great asset in further developing EEROS in a relatively forgiving academic environment. Additionally, we have expressed interest in contributing to or using EEROS, and we plan to be active members of the EEROS community upon completion of this project.

## Chapter 5: Recommendations and Conclusions

We now look to provide the EEROS developers with recommendations as they continue their work. These suggestions are directly related to our original objectives. Our findings have shown that these recommendations should not be taken as long term plans, but rather a means to continuously improve as the EEROS community expands, which aligns with the Agile mentality. It is important to assess the current state of EEROS, and try to understand what kind of challenges and issues the EEROS team will face in the immediate future. Finally, we step back and examine our work with EEROS with a global perspective. These larger themes are the means to understanding the greater impact that our project has had. These conclusions also help to encapsulate what it means to be a new open source project, both in the potential that comes with such a venture and the challenges that a project will face at this stage.

### 5.1 EEROS Presence

The EEROS developers should take several steps to ensure that EEROS continues to have a thriving web presence. First and foremost, the website should be kept up to date. An out of date website can be a huge detriment to any open source project; it can appear that the project is no longer actively developed and supported. Continually updating the information on the website will make EEROS more attractive, since potential users will know that the developer team is devoted to helping the user base and improving their experience.

Most community interaction will take place on the EEROS blog page, on social media sites, or simply over E-mail. Public forums like the blog or social media are aimed primarily at users, but strong developer interaction can help these places become more helpful. Therefore, we recommend that the developer team takes the lead in promoting these tools by using the blog and social media to promote new features and improvements. They should interact with the community as much as possible. Helping users work around problems that they discover not only reveals new issues and bugs, but also promotes EEROS as well-supported software. This kind of user feedback is valuable and helpful to developers.

Another area that can benefit from community contribution is the Wiki. Traditionally, Wikis are written by a community and they can be edited by users in that community. This doesn't mean that the Wiki should be free for anyone to edit, but allowing EEROS users to submit content will be helpful in allowing EEROS to grow. It can be worrisome allowing others to change crucial information. It requires trust that they will not harm the page in some way or provide false information. However, open source communities are in general very appreciative of this trust, and as a community grows it regulates information and changes better than a dedicated team could. In the long run, this change will be an important step in allowing EEROS to be truly community-developed, marking a transition for the EEROS team from developing code to developing a culture in the robotics community around robotics software being Easy, Elegant, Reliable, Open, and Safe.

## 5.2 Sustainable Funding

The EEROS team already has a good track record of maintaining funding for EEROS. By applying for funding applications from a variety of European sponsors, they have been able to maintain the development of EEROS. This has worked well for them, and so they should continue using funding applications until a better method is required. However, many of these applications are written in English, and since the EEROS team's native language is German, phrases and grammar don't always translate well. Proper proofreading for English grammar mistakes can go a long way to getting a funding application approved, and taking the time to have a native speaker proofread their applications will be a worthwhile investment.

Once EEROS has developed a much larger user base, crowdfunding and donations may prove to be viable sources for continued, sustainable funding. Setting up a way for users to donate to the EEROS team is a good idea as well, but will not likely bring in significant funds at this stage of development. Crowdfunding may prove useful if the EEROS team requires additional funding for a new, larger project surrounding EEROS. Websites such as Kickstarter and Indiegogo are excellent examples of crowdfunding websites that the EEROS team should look into. They could also provide funding for new EEDURO robots that will expand EEROS's user base.

## 5.3 Streamlined Development

So far, the core EEROS developers have created a working version of their software. However, it would be helpful for the team to start implementing important development practices before the project grows any larger. The team should decide on standards for coding, software testing, Git (version control), and documentation. The resources we have provided should give the team a good starting point to initially develop and adapt these guidelines for themselves.

One obstacle to the expansion of EEROS right now is the lack of documentation. While the Wiki contains some good tutorials, the EEROS API documentation isn't as well developed. We recommend that the team start the process of streamlining their development by creating standards for documentation and ensuring that contributions to EEROS meet the documentation standards before they are accepted. Good documentation will allow new EEROS users to use the framework without needing to be in direct communication with the core developers.

Since the EEROS team eventually wants EEROS to become community-developed software, it is important that the team comes up with a process for using Git. Currently, the EEROS GitHub is not very active and changes are rarely committed. As development continues, there should be a process for committing new changes to EEROS. Our annotated bibliography contains some resources on proper Git usage, and we recommend that the EEROS team start following a Git branching model.

Once the documentation and Git process have been defined, EEROS will be ready for a general use release. It will also be in a good position to start accepting contributions from the community rather than just the core team. Once the software is released, the team will start

receiving feedback, which will allow the Agile development model to show its strengths. We recommend using Agile development because we have found that it works well in practice, not just because it has become a standard in software development. If the development model chosen is not working, it would be beneficial to adapt to another model. We have provided resources on various Agile methods, and hopefully one or more of these will be helpful to the EEROS team.

## **5.4 Healthy Community**

As mentioned earlier in our paper, it is important for the EEROS team to monitor the community surrounding their project, as much of their success depends on the community. We developed an analysis tool for the EEROS team to analyze how the members of the development community interact when writing code. This information will be valuable to the EEROS team so that they can understand what's going on in the project, but not by itself. To make the information that the tool provides useful, the EEROS team will need to use information about development practices that we present in this paper and their own experience to both interpret the information that the tool provides, and determine how to fix it. For example, they might ask why two or three developers seem to be counterproductive when they're working on a part of the code that constantly is rewritten, or understanding that a sudden influx of new developers may lower how responsive the developers are to different problems because the new developers aren't acclimated to the community yet.

The EEROS developers should remember the limitations of our analysis as well. It only considers interaction between developers in regards to writing code. There are a lot of practices that are important to the development community that can only be measured individually, such as a certain developer's habits in regard to coding guidelines, or a general adherence to release schedules and best practices. When working on a software project, there is a tendency to stop following these practices to try and finish critical work when a deadline approaches. While this may occur, the EEROS team needs to promote certain expectations among its developers in an effort to cultivate a healthy environment.

Finally, the EEROS team should keep in mind that the health of their community isn't only measured by how the developers interact through writing code. EEROS aims to serve a much larger community where many users will never contribute to the code, but where the usage will inform the development. As important as it is to monitor the developers, the EEROS team needs to also monitor the website and other social media tools. This can be done through the analytical tools that we installed through WordPress for the website, but also by keeping an eye on how active users are on the social media platforms the EEROS teams are using.

These three areas are important for EEROS to continue to keep track of if they want to make sure that they cultivate a healthy community that will make their framework a success.

## 5.5 New Partners

The developers are currently looking for new partners to reach out to. We worked together to create an annotated list of local and international partners, but we also want to make recommendations about when to reach out to the different types of partners. We believe that the developers should first look to educational partners. These will provide a solid group of test users for EEROS, and some of them will become the first developers not at NTB. From here, EEROS will be used in research projects, further testing and developing its capabilities. This step will allow EEROS to become stable enough for industrial use. Industrial partners should come last, since they need software that is fully developed and completely stable. EEROS will get to that point after it has seen widespread use in the educational world. This is a guideline, not an absolute rule. If industrial partners want to try EEROS now, they should, providing that they understand that all features may not be implemented. All new partners are good, and will provide useful feedback in some form (even rejection of EEROS can be useful). However, the EEROS team should focus *their* efforts to find new partners in the provided order to have the most impact moving forward.

We believe that WPI could be an extremely effective partner, especially at this early stage. As mentioned before, WPI's MQP program is a great platform to test and further develop EEROS. By establishing contacts at WPI, the EEROS developers will gain a valuable foothold in educational robotics in the United States. Having this beginning of a global community will allow EEROS to expand at a much more rapid rate. We recommend that the EEROS developers take full advantage of these contacts, and continue to build a relationship between NTB and WPI by developing EEROS. This way, EEROS has the chance to succeed overseas as well as domestically, and it will serve to increase the impact that the software will have on the robotics community.

## 5.6 Conclusions

Over the course of the project, the EEROS team has mentioned that they want to move towards an Agile development model, which is a way of developing software where the focus is adapting to change rather than following a plan. The point of the Agile development model is that the developers constantly check in with the customer or user to make sure that the software is useful to the user. This will be very important for EEROS as they may have a diverse user base. The EEROS team will have to work with users to ensure the framework is flexible enough for all users. The open source model that the EEROS team plans to follow is a way to accomplish this; in the open source model of development, the users have direct feedback into how the framework will be developed and how it will serve their needs. If EEROS is able to establish a community presence early enough, they will be well poised to achieve their goals.

One of the things we learned when we got to Switzerland was that EEROS was not the only one that benefitted from an Agile development model. Our original approach to the project was to provide long term planning and resources that would support the development team's

eventual expansion two or three years after we completed our project. This was the wrong approach; we were far more successful by constantly checking in with the development team and providing short term tools that the EEROS team could use to create their own long term plans. Then they could adapt to unexpected problems that may end up occurring later in the development process. This taught us that in any project the Agile development model can be applied to create a better working relationship. The Agile model was originally designed for software development, which can usually be seen as a project; a group of developers working towards a common goal. We found that the development model that we suggested to the developers could be applied quite effectively to describe our own processes, and that we should interact with them just as they would hope to interact with their users.

We have made a point in this paper to describe how an open source project needs a community to thrive. In a way, community development can be seen as a form of Agile development. The community works constantly on short term implementation and creating working code, and they can respond to user feedback very quickly. The users and developers can simply rely on a feature's popularity to assess its importance to the project. In this way, we have discovered that the very first research we completed on open source software and communities (section 2.1) directly relates to this realization that we had at the end of our project. Perhaps the parallel between open source community interaction and our interaction with the EEROS developers is not a coincidence. What we've found (and what may be at the heart of the open source movement in general) is that the success of a project can depend more on the interaction of its creators and end users than the creator's ability to perceive the project's objectives. This interaction provides flexibility, which is a powerful quality that is vital to a project's success. As we've observed from working with the EEROS team, young open source projects naturally have this kind of flexibility; the mark of a successful open source project is the ability to hold on to this flexibility while transitioning into a mature project.

## Bibliography

Athey, S., & Ellison, G. (2014). Dynamics of open source movements. *Journal of Economics & Management Strategy*, 23(2), 294-316. doi:10.1111/jems.12053

Barham, A. (2013). The Emergence of Quality Assurance Practices in Free/Libre Open Source Software: A Case Study. *IFIP Advances in Information and Communication Technology*, 404(21), 271-276. Retrieved February 15, 2015.

Beck, K. et al. (2001). Manifesto for Agile Software Development. Retrieved April 23, 2015 from <http://www.agilemanifesto.org>

Blondel, V. D., Guillaume, J., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10), P10008. doi:10.1088/1742-5468/2008/10/P10008

Bonaccorsi, A., & Rossi, C. (2003). Why open source software can succeed. *Research Policy*, 32(7), 1243-1258. doi:10.1016/S0048-7333(03)00051-9

Carillo, K., & Okoli, C. (2008). THE OPEN SOURCE MOVEMENT: A REVOLUTION IN SOFTWARE DEVELOPMENT. *The Journal of Computer Information Systems*, 49(2), 1.

Collins-Sussman, Brian W. Fitzpatrick. Ben, & Safari Books Online. (2012). *Team geek* O'Reilly Media, Inc.

Company profile. (2015). 2015, from <http://www.redhat.com/en/about/company>

Corbet, J., Kroah-Hartman, G., & McPherson, A. (2013). Linux Kernel Development: How Fast It is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It.

Dorsey, M. (2011). WPI Recieves Accreditation for First-in-the-Nation Robotics Engineering Undergraduate Degree Program. *Worcester Polytechnic Insitiute*. Retrieved April 23, 2015 from <http://www.wpi.edu/news/20112/rbeaccred.html>

Fitzgerald, B. (2006). The Transformation of Open Source Software. *MIS Quarterly*, 30(3), 587-598.

Form 10-K 2014. Red Hat, Inc.

Glass, R. L. (1999). The loyal opposition of open source, linux...and hype. *IEEE Software*, 16(1), 128, 126-127. doi:10.1109/52.744583

Gostai: Urbi. (2015). Retrieved January 29, 2015, from <http://www.gostai.com/products/urbi/>

Grinzo, L. & Fernandez, L. (1999). Clarifying the Open Source movement (Vol. 24, pp. 119). San Mateo: UBM LLC.

Guhl, T., & Bischoff, R. (2009). Robotic Visions to 2020 and beyond – The Strategic Research Agenda for robotics in Europe. Retrieved 2/15/2015, from [http://robotics.h2214467.stratoserver.net/cms/upload/SRA/2010-06\\_SRA\\_A4\\_low.pdf](http://robotics.h2214467.stratoserver.net/cms/upload/SRA/2010-06_SRA_A4_low.pdf)

Harrison, A. (2013). Rise of the New Geeks: How the Outsiders Won. *The Guardian*. Retrieved from <http://www.theguardian.com/fashion/2013/sep/02/rise-geeks-outsiders-superhero-movies-dork>

Hodges, A. Alan Turing: The Enigma. Retrieved February 22nd, 2015, from <http://www.turing.org.uk/index.html>

Jalopy, M. (2005). Owner's Manifesto. *Make*.

Krishnamurthy, S. (2005). An analysis of open source business models.

Kelly, Devin W. Student author -- ECE, Holtorf, Gregory K. Student author -- CS, & Rissmiller, Kent J. Faculty advisor -- SS. (2009). *The affects of open source software licenses on business software*. Worcester, MA: Worcester Polytechnic Institute.

Lerner, J., & Tirole, J. (2001). The open source movement: key research questions. *European economic review*, 45(4/6), 819-826. doi: 10.1016/S0014-2921(01)00124-6

Licenses - GNU Project - Free Software Foundation. (n.d.). Retrieved February 16, 2015, from <http://www.gnu.org/licenses/licenses.html>

Orocos Overview. (2007). Retrieved January 29, 2015, from <http://www.people.mech.kuleuven.be/~orocos/pub/documentation/rtt/current/doc-xml/orocos-overview.html>

Phillips, D. E., & Ebrary Academic Complete. (2009). *The software license unveiled: How legislation by license controls software access*. Oxford: Oxford University Press.

Raymond, E. S., & eBooks on, E. (1999). *The cathedral & the bazaar: musings on Linux and open source by an accidental revolutionary*. Cambridge, Mass: O'Reilly.

Red Hat Patent Policy. (2014). from [http://www.redhat.com/legal/patent\\_policy.html](http://www.redhat.com/legal/patent_policy.html)

ROS.org History. (2015). Retrieved January 29, 2015, from <http://www.ros.org/history>

ROS-Industrial. (2015). Retrieved January 29, 2015, from <http://www.rosindustrial.org>

ROS Introduction. (2014). Retrieved January 29, 2015, from <http://www.wiki.ros.org/ROS/Introduction>

ROS.org Powering the world's robots. (2015). Retrieved January 29, 2015, from <http://www.ros.org>

ROS Wiki. (2014). Retrieved January 29, 2015, from <http://www.wiki.ros.org>

Stallman, R. (2009). Why "Open Source" Misses the Point of Free Software (Vol. 52, pp. 31). New York: Association for Computing Machinery.

Sargent, G. (2015). Robotics History Timeline. from [http://robotics.ece.auckland.ac.nz/index.php?option=com\\_content&task=view&id=31](http://robotics.ece.auckland.ac.nz/index.php?option=com_content&task=view&id=31)

Silic, M., & Back, A. (2013). Information Security and Open Source Dual Use Security Software: Trust Paradox. *IFIP Advances in Information and Communication Technology*, 404(21), 194-206. Retrieved February 15, 2015.

Spencer, R. (2002). Got a dirty, dangerous, dull job? let a robot do it and keep your workers safe. *Robotics World*, 20(2), 14-15. Retrieved from <http://ezproxy.wpi.edu/login?url=http://search.proquest.com/docview/218467975?accountid=29120>

St. Amant, K., & Still, B. (2007). *Handbook of research on open source software technological, economic, and social perspectives* (pp. xxxvi, 728 p. ill. 729 cm.). Retrieved from <http://AU4SB9AX7M.search.serialssolutions.com/?V=1.0&L=AU4SB9AX7M&S=JCs&C=TC000080965&T=marc&tab=BOOKS>

The Player Project. (2014). Retrieved January 29, 2015, from <http://www.playerstage.sourceforge.net>

The Orocos Project. (2015). Retrieved January 29, 2015, from <http://www.orocos.org>

Unimate. (2003). Retrieved February 22nd, 2015, from <http://www.robothalloffame.org/inductees/03inductees/unimate.html>

Vaughan-Nichols, S. (2015). Red Hat: The first billion dollar Linux company has arrived | ZDNet. from <http://www.zdnet.com/article/red-hat-the-first-billion-dollar-linux-company-has-arrived/>

Williams, S. (2004). *Free as in Freedom: Richard Stallman's Crusade for Free Software*: Project Gutenberg U6 - ctx\_ver=Z39.88-2004&ctx\_enc=info%3Aofi%2Fenc%3AUTF-8&rft\_id=info:sid/summon.serialssolutions.com&rft\_val\_fmt=info:ofi/fmt:kev:mtx:book&rft.genre=book&rft.title=Free+as+in+Freedom&rft.au=Williams%2C+Sam&rft.date=2004-01-01&rft.pub=Project+Gutenberg&rft.externalDocID=1186922&paramdict=en-US U7 - eBook U8 - FETCH-wpi\_catalog\_11869223.

Young, R. (1999). Giving It Away: How Red Hat Software Stumbled Across a New Economic Model and Helped Improve an Industry. *Open Sources: Voices from the Open Source Revolution*.

## Appendix A: Interview Notes

The following are our raw notes from interviews with Urs Graf and Adam Bajric, two of the first developers of EEROS. Urs has been responsible for the software architecture, whereas Adam has been one of the lead programmers.

Table 2: Interview Notes

What has your role been in EEROS development?	
Urs	<ul style="list-style-type: none"><li>- 5 years or so ago</li><li>- Wrote first version in Java<ul style="list-style-type: none"><li>- Like Simulink</li></ul></li><li>- Masters student wrote first full Java version</li><li>- Been with the team since the beginning</li><li>- Coding and lots of reviews, lots of software architecture</li></ul>
Adam	<ul style="list-style-type: none"><li>- Working on it since the beginning</li><li>- Made math library in framework</li><li>- Made sequencer</li><li>- Parts of control system</li></ul>

What is good about EEROS?	
Urs	<ul style="list-style-type: none"><li>- Framework that is flexible but has a rigid safety system</li><li>- Safety system is the selling point</li><li>- Blocks and time domains (still needs implementations)</li></ul>
Adam	<ul style="list-style-type: none"><li>- Nice interface: biggest feature</li><li>- Easy to use</li><li>- Interface improved from first prototype a lot</li></ul>

Favorite and least favorite feature?	
Urs	<ul style="list-style-type: none"><li>- Needs profiling features (logging of real-time tasks)</li></ul>
Adam	<ul style="list-style-type: none"><li>- The subsystem is not completed.</li><li>- The safety system has not been rewritten and improved</li></ul>

<b>How does the EEROS development model work? Is there one maintainer? Is Git used properly? etc.</b>
---

Urs	<ul style="list-style-type: none"> <li>- No real development model yet</li> <li>- Needs to be defined</li> <li>- Not using Git flow yet</li> </ul>
Adam	<ul style="list-style-type: none"> <li>- Use Git and GitHub and sometimes reviews with pull requests but usually just look at problems together</li> <li>- No</li> </ul>

#### **How is communication between developers? Mailing list?**

Urs	<ul style="list-style-type: none"> <li>- Meetings in the lab</li> <li>- Mailing list with people from Winterthur</li> <li>- Conference calls</li> </ul>
Adam	<ul style="list-style-type: none"> <li>- Just talk to each other</li> </ul>

#### **Are you looking for EEROS to become community-developed? Are you looking for more developers in general?**

Urs	<ul style="list-style-type: none"> <li>- Goal is to be community-developed</li> <li>- Problem is rigid structure right now</li> <li>- Conflict between rigid structure yet invite new ideas</li> <li>- The safety system makes it hard for there to be many developers working on the core system</li> <li>- Kernel contribution is difficult, extensions will be easier</li> </ul>
Adam	<ul style="list-style-type: none"> <li>- Yes, Development process is important.</li> <li>- Would like to see people in industry contributing and using it.</li> <li>- Only students using it now</li> <li>- Will establish itself better if industry uses it</li> </ul>

#### **As a developer, how easy do you think it is to use EEROS? Good documentation? Simple API?**

Urs	<ul style="list-style-type: none"> <li>- Wrote most of the information on the Wiki</li> <li>- API has gone through many iterations</li> <li>- Times when the API is clunky</li> </ul>
Adam	<ul style="list-style-type: none"> <li>- Documentation not very good</li> <li>- New developer would need effort to get started</li> <li>- Too time consuming to work on it</li> <li>- EEROS changes a lot and documentation would have to keep changing</li> </ul>

**What do you think about the EEROS community so far?**

Urs	- Too small to make a real judgment
-----	-------------------------------------

Adam	- Not very big
------	----------------

**What are your hopes for the community and EEROS in the future?**

Urs	<ul style="list-style-type: none"><li>- Hoping that next release will be stable enough for people to use it</li><li>- Documentation still not good enough</li><li>- Layer in between OS and EEROS not finished</li><li>- More hardware</li></ul>
-----	--

Adam	- N/A
------	-------

**What coding style (if any) do you follow? If so, is it established?**

Urs	- Try to follow the Google C++ style with some modifications
-----	--

Adam	- Google coding guidelines modified
------	-------------------------------------

**Are there plans for an easier installation process (deb package, etc...)?**

Urs	<ul style="list-style-type: none"><li>- Eventually, but not yet</li><li>- External company to create packages</li></ul>
-----	---

Adam	- Thought about making Debian package, didn't do it
------	---

**Do you have a release process / is anyone in charge?**

Urs	<ul style="list-style-type: none"><li>- Not yet</li><li>- Established in another two open source projects</li></ul>
-----	---

Adam	<ul style="list-style-type: none"><li>- Not really</li><li>- Would help to have one</li><li>- Different for each platform</li></ul>
------	---

**How prominent is unit testing in your process of development?**

Urs	<ul style="list-style-type: none"><li>- JUnit for EEROS Java</li><li>- Looking into automatic testing for C++</li></ul>
Adam	<ul style="list-style-type: none"><li>- Unit tests are general</li><li>- Not priority</li><li>- Didn't start with test driven development, so didn't pick it up</li></ul>

## Appendix B: Before and After Screenshots of EEROS Homepage

### Before:



Figure 9: Old EEROS Homepage

After:



Figure 10: New EEROS Homepage

To see more, visit <http://www.eeros.org>.

## Appendix C: Website Screenshots



Figure 11: "What is EEROS?" page

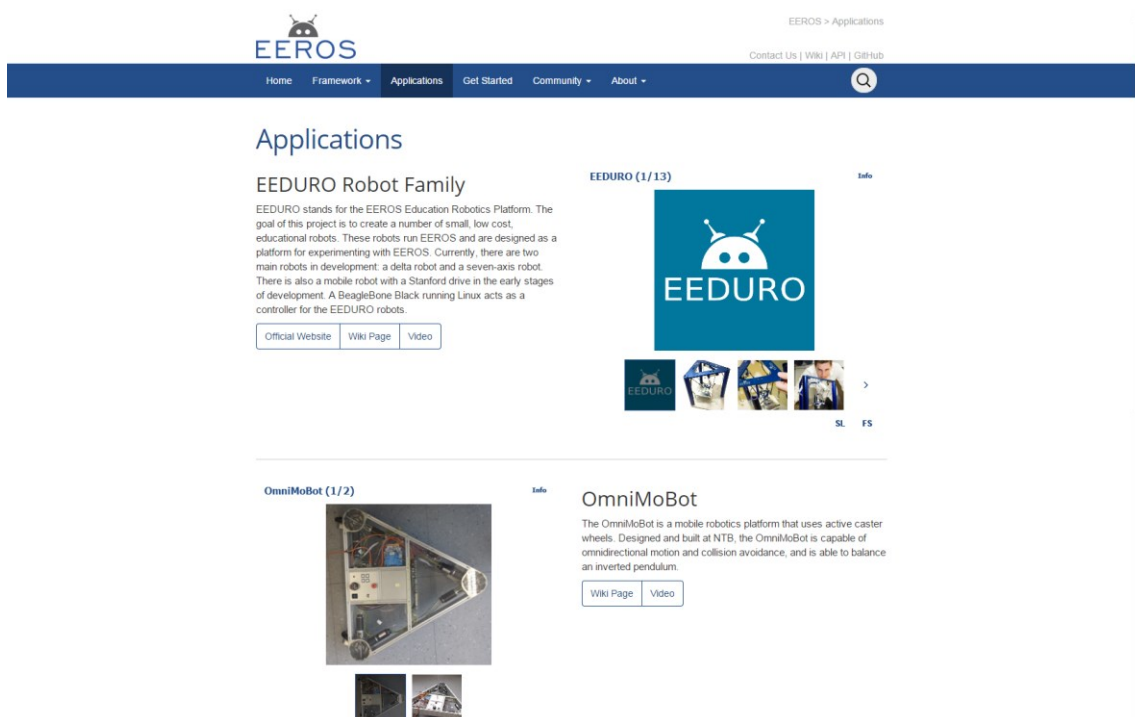



Figure 12: "Applications" page



EEROS > Get Involved

[Contact Us](#) | [Wiki](#) | [API](#) | [GitHub](#)

[Home](#) | [Framework](#) | [Applications](#) | [Get Started](#) | [Community](#) | [About](#)

## Get Involved

### Contribute

EEROS is currently being developed solely in-house. Check here later for info on when EEROS will be open for public contribution.

Feel free to [contact us](#) with any suggestions for further improvements to EEROS.

---

### Stay Updated

If you'd like to stay informed about further developments, please subscribe to our mailing list on the right.

### Subscribe

\* Indicates a required field

First \*

Last \*

Email \*

Use \*

Commercial
Educational
Private

Organization

Visibility \*

Confidential
Members Only
Public

[Additional Information](#)

☒ Receive EEROS-related news via email

Human Verification \*  + 3 = 5

Figure 13: “Get Involved” page



EEROS > The EEROS Team

[Contact Us](#) | [Wiki](#) | [API](#) | [GitHub](#)

[Home](#) | [Framework](#) | [Applications](#) | [Get Started](#) | [Community](#) | [About](#)

## The EEROS Team



### Einar Nielsen

EEROS Project Manager

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam commodo hendrerit nunc a pretium. Nunc volutpat sagittis lorem, sit amet molestie tellus lobortis nec. Mauris tempus urna du, sed ultricies tortor sagittis et. Nam ultricies elit eu risus molestie, ac rutrum lectus lobortis.



### Urs Graf

EEROS Developer

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam commodo hendrerit nunc a pretium. Nunc volutpat sagittis lorem, sit amet molestie tellus lobortis nec. Mauris tempus urna du, sed ultricies tortor sagittis et. Nam ultricies elit eu risus molestie, ac rutrum lectus lobortis.



### Martin Züger

EEROS Developer

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam commodo hendrerit nunc a pretium. Nunc volutpat sagittis lorem, sit amet molestie tellus lobortis nec. Mauris tempus urna du, sed ultricies tortor sagittis et. Nam ultricies elit eu risus molestie, ac rutrum lectus lobortis.



###



###



###

Figure 14: “The EEROS Team” page

63

## Appendix D: Community Analysis Math

The following short document explains the math behind our community analysis tool. The documentation was written with LaTeX and designed to be viewed as a PDF. You can see the original document with the other project documents.

### Graph Theoretical Analysis of Developer Communities

Nathan Hughes

April 27, 2015

#### 1 Introduction

This document outlines how the WPI EEROS IQP Analysis Tool generates a network model of developers from a software project's code.

#### 2 Input Data

The tool takes the commit tree structure of the main repository associated with a software project and combines it with other forks of the software project, eventually creating a commit tree that contains every commit ever made that's publically available that results from the first commit of the main repository. Once the commit tree is created, it's updated using the all the file differences between consecutive commits.

#### 3 Classification of Interaction

The actual network is built up using by using file differences between consecutive commits. The commit tree structure is traversed in a modified breadth-first order where tie breaking between commits is done by the commit date. Each commit's file differences are used to calculate an interaction strength between the current commit and previous commits. An interaction strength between two developers is given by a weighted sum of the number of strong interactions and weak interactions. A strong interaction is when two

developers edit the same section of code in consecutive commits. A weak interaction is when two developers edit the same file within a certain timespan. The interaction strength for a given commit and a given pair of developer is given by:

$$s_c(u, v) = 0.5 * n_s(u, v) + 0.1 * n_w(u, v)$$

where  $s_c(u, v)$  is the interaction strength between developer  $u$  and developer  $v$ ,  $n_s(u, v)$  is the nubmer of interactions between developer  $u$  and developer  $v$  and  $n_w(u, v)$  is the number of interactions between developers  $u$  and  $v$ .

The strength of a relationship between developer  $u$  and  $v$  is given by:

$$s(u, v) = s_c(u, v) + t \left( \sum_{\forall c' \in C | c' < c} (s_c(u, v)), c \right)$$

where  $C$  is the collection of all the commits, and  $t(x, y)$  is a function used to decay the value of a relationship over time.

The weight of each edge in a relationship is given by  $s(u, v)$  and edges only exist between  $u$  and  $v$  when  $s(u, v) > 0.5$ .

## 4 Analysis of Developer Network

Using the network of developers, it is possible to derive the estrada index and average closeness of the the network.

To calculate the estrada index, we first calculate the communicability of the developer network. This is given by:

$$G = e^w$$

where  $w$  is the weighted adjacency matrix of the graph.

The estrada index is calculated as follows:

$$\ln \left( \sum_{v \in V} G_{vv} \right)$$

where  $V$  is the set of all the developers

To calculate the average closeness, we first calculate the closeness for each developer, which is given by:

$$g(u) = \frac{1}{\sum_{v \in V} d(u, v)}$$

where  $d(u, v)$  is the distance between developers  $u$  and  $v$  where the distance of each edge is the inverse of it's weight.

The average closeness is then given by:

$$\frac{\sum_{v \in V} g(v)}{|V|}$$

## Appendix E: Development Resources Annotated Bibliography

Table 3: Development Resources Annotated Bibliography

Title	Description	Media	URL
Agile Manifesto	<ul style="list-style-type: none"> <li>- Describes founding principles for Agile Development</li> </ul>	Website	<a href="http://agilemanifesto.org/">http://agilemanifesto.org/</a>
DACS State-of-the-Art	<ul style="list-style-type: none"> <li>- History of different development methods</li> <li>- Discusses Agile and criteria for when Agile should be adopted</li> <li>- Discusses various Agile methods: XP, Scrum, Crystal Methods, Feature-Driven Development, Lean, Dynamic Systems Development</li> </ul>	Website	<a href="http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.201.2704&amp;rep=rep1&amp;type=pdf">http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.201.2704&amp;rep=rep1&amp;type=pdf</a>
Handbook of Research on Open Source Software	<ul style="list-style-type: none"> <li>- Excellent compilation of academic papers on everything open source, from the social aspects to development models to licensing and more</li> </ul>	Book (180.00 USD, but free PDF can be found)	<a href="http://www.igi-global.com/book/handbook-research-open-source-software/494">http://www.igi-global.com/book/handbook-research-open-source-software/494</a>
Two case studies of open source software development	<ul style="list-style-type: none"> <li>- Case studies for Mozilla and Apache Open source development</li> <li>- Look at users and developers contributions over time in relation to success of project</li> <li>- Look at contributions via email and new code creation, defect fixing, etc.</li> <li>- How do open source developers identify and solve problems with no real timelines</li> </ul>	Academic Research Paper	<a href="http://dl.acm.org/citation.cfm?id=567795">http://dl.acm.org/citation.cfm?id=567795</a>
Quality assurance under the open source development model	<ul style="list-style-type: none"> <li>- How is software QA performed under open source</li> <li>- How it differs from traditional development models</li> <li>- Can these differences give practical advantages</li> <li>- Findings: still developing (world of open source) and it isn't advantageous in all scenarios</li> </ul>	Academic Research Paper	<a href="http://www.science-direct.com/science/article/pii/S016412120200064X">http://www.science-direct.com/science/article/pii/S016412120200064X</a>
FreeBSD Project Case Study	<ul style="list-style-type: none"> <li>- Compares FreeBSD to Apache to look at open source success</li> <li>- FreeBSD has 1) smaller set of core developers 2) larger set of top developers 3) more well defined testing process</li> <li>- Since Apache and FreeBSD are both successful, perhaps these differences won't make or break success</li> <li>- Both systems have similar ratio of</li> </ul>	Academic Research Paper	<a href="http://ieeexplore.ieee.org/xpl/login.jsp?tp=&amp;arnumber=1463231&amp;url=http%3A%2F%2Fieeexplore.ieee.org%2Fexpls%2Fabs_all.jsp%3Farnumber%3D1463231">http://ieeexplore.ieee.org/xpl/login.jsp?tp=&amp;arnumber=1463231&amp;url=http%3A%2F%2Fieeexplore.ieee.org%2Fexpls%2Fabs_all.jsp%3Farnumber%3D1463231</a>

	core developers to debuggers, defect densities, and developers who are users		
The Impact of Ideology on Effectiveness in Open Source Software Development Teams	<ul style="list-style-type: none"> <li>- Framework of OSS community ideology</li> <li>- Theoretical model to show how adherence to different parts of framework affects success</li> <li>- Success defined as retention of developer input and generation of project outputs</li> <li>- Adherence to some parts of ideology can increase retention but decrease outputs</li> </ul>	Academic Research Paper	<a href="http://www.jstor.org/discover/10.2307/25148732?uid=2&amp;uid=4&amp;sid=21106140420621">http://www.jstor.org/discover/10.2307/25148732?uid=2&amp;uid=4&amp;sid=21106140420621</a>
Team Geek	<ul style="list-style-type: none"> <li>- Look at patterns and anti-patterns for working with other people developing software</li> <li>- Human component of software engineering</li> <li>- Learning to collaborate and have soft skills will let you have more impact for same effort</li> </ul>	Book (25.00 USD)	<a href="http://shop.oreilly.com/product/0636920018025.do">http://shop.oreilly.com/product/0636920018025.do</a>
Software Documentation	<ul style="list-style-type: none"> <li>- Suggested format for good documentation</li> <li>- Format, writing style, guidelines, standards, etc.</li> </ul>	PDF	<a href="http://www.literateprogramming.com/documentation.pdf">http://www.literateprogramming.com/documentation.pdf</a>
Agile/Lean Documentation	<ul style="list-style-type: none"> <li>- Begins with explaining why documentation is important</li> <li>- What kinds of documents should be created for agile development</li> <li>- Handoffs, updates, other best practices for documentation upkeep</li> </ul>	Website	<a href="http://www.agilemodeling.com/essays/agileDocumentation.htm">http://www.agilemodeling.com/essays/agileDocumentation.htm</a>
Software Testing: A Craftman's Approach	<ul style="list-style-type: none"> <li>- Comprehensive book about software testing</li> <li>- Covers mathematical models, unit testing, software reviews, etc.</li> <li>- Has sections specifically for testing in an Agile environment</li> </ul>	Book (about 90.00 USD)	<a href="http://www.crcpress.com/product/isbn/9781466560680">http://www.crcpress.com/product/isbn/9781466560680</a>
Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation	<ul style="list-style-type: none"> <li>- Book about automating as many parts of the 'deployment pipeline' as possible</li> <li>- Seems to be fairly comprehensive</li> </ul>	Book (about 40.00 USD)	<a href="http://www.amazon.com/Continuous-Delivery-Deployment-Automation-Addison-Wesley/dp/0321601912">http://www.amazon.com/Continuous-Delivery-Deployment-Automation-Addison-Wesley/dp/0321601912</a>

## Appendix F: Annotated List of Partners

Table 4: Annotated List of Partners

Name	Organization	Type	Description	Focus
Scott Barton	WPI	Professor	Prof. Barton is a music professor at WPI who runs the music perception and robotics laboratory. He's interested in robust, real time software as a way to both synchronize many robots, and to process audio input	Artistic Robotics
Michael Gennert	WPI	Professor	Head of the robotics department at WPI. Not very active in student projects	Education
Nathan Hughes	WPI	Student	Student at WPI, research interests are motion planning and graph theory.	Motion Planning, Musical Robotics
Dmitry Berenson	WPI	Professor	Prof. Berenson works with human robot interaction and motion planning. Primarily uses the PR2 and Baxter robot as research platforms.	Motion Planning, Autonomous Collaboration
Susan Jarvis	WPI	Professor	Prof. Jarvis teaches the embedded computing courses at WPI. Her research focus on sensing using robotic platforms	Embedded Computing, Sensing
Jim Duckworth	WPI	Professor	Prof. Duckworth teaches advanced FPGA courses at WPI. His research focus is real time systems, and he sponsors several projects dealing with real time sensing. He also works a lot with industry	Sensing, Real Time Systems
Craig Putnam	WPI	Professor	Prof. Putnam teaches the introductory level courses at WPI, and the manufacturing courses. He doesn't do research, but sponsors a lot of student projects, especially the projects that involve the control of industrial arms	Manufacturing
Creative Robotics Studio	WPI	Project	Project investigating how to create a framework for controlling artistic robots (IQP & MQP)	Artistic Robotics