Technology Innovation Management Review

January 2013

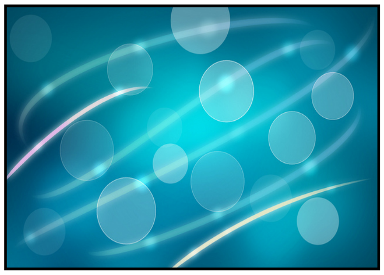


Image licensed under CC BY by Jeffpro57

Open Source Sustainability

Welcome to the January 2013 issue of the *Technology Innovation Management Review*. The editorial theme of this issue is Open Source Sustainability. We invite your comments on the articles in this issue as well as suggestions for future article topics and issue themes.

Carleton University

www.timreview.ca

Publisher

The *Technology Innovation Management Review* is a monthly publication of the Talent First Network.

ISSN

1927-0321

Editor-in-Chief

Chris McPhee

Overview

The *Technology Innovation Management Review* (TIM Review) provides insights about the issues and emerging trends relevant to launching and growing technology businesses. The TIM Review focuses on the theories, strategies, and tools that help small and large technology companies succeed.

Our readers are looking for practical ideas they can apply within their own organizations. The TIM Review brings together diverse viewpoints – from academics, entrepreneurs, companies of all sizes, the public sector, the community sector, and others – to bridge the gap between theory and practice. In particular, we focus on the topics of technology and global entrepreneurship in small and large companies.

We welcome input from readers into upcoming themes. Please visit timreview.ca to suggest themes and nominate authors and guest editors.

Contribute

Contribute to the TIM Review in the following ways:

- Read and comment on past articles and blog posts.
- Review the upcoming themes and tell us what topics you would like to see covered.
- Write an article for a future issue; see the author guidelines and editorial process for details.
- Recommend colleagues as authors or guest editors.
- Give feedback on the website or any other aspect of this publication.
- Sponsor or advertise in the TIM Review.
- Tell a friend or colleague about the TIM Review.

Please contact the Editor if you have any questions or comments: timreview.ca/contact

Editorial: Open Source Sustainability

Chris McPhee, Editor-in-Chief
Maha Shaikh, Guest Editor

From the Editor-in-Chief

Welcome to the January 2013 issue of the *Technology Innovation Management Review*. This month's editorial theme is Open Source Sustainability. It is my pleasure to welcome our guest editor for this issue, **Maha Shaikh**, Assistant Professor of Information Systems at Warwick Business School in the United Kingdom, who has assembled a diverse line up of authors to offer their perspectives on the sustainability and governance of open source software.

As always, we welcome your feedback, articles, and suggestions for future themes. We hope you enjoy this issue of the TIM Review and will share your comments online. Please also feel free to contact us (timreview.ca/contact) directly with feedback or article submissions.

Chris McPhee
Editor-in-Chief

From the Guest Editor

The theme of this issue was triggered by a discussion with Daniel Curto-Millet (a doctoral student and one of the authors in this issue), who is particularly interested in Elinor Ostrom's work in relation to sustainability (tinyurl.com/pcxroc) and how it is applicable to open source software. My own research more recently has made me very curious about the dimensions and conditions necessary to sustain an open source community, project, and ecosystem.

The idea of sustainability, though borrowed from natural resource management, is surprisingly applicable to open source ecosystem sustainability. The definition of sustainability that resonated the most with my understanding of open source was provided by Repetto (1986; tinyurl.com/afrmww9), and I have amended it slightly to make it sensible for open source:

Open source sustainability is the recognition and drive to manage all assets, and resources related to open source development, including the broader financial and physical assets in order to increase the long-term vibrancy and well-being of a project (and ecosystem). Sustainable development of open source, as a goal, rejects policies and practices that support current adoption and development in the short-term without regard for how this may deplete the productive base, including all resources, and that leaves future communities with poorer prospects.

As this definition implies, time is a dimension that causes fluctuations in what is sustainable and desirable in open source. Each open source project has its own lifecycle (Schweik, this issue) though, of course, some never see growth and are simply abandoned. Is abandonment caused by a depletion of the productive base? The inability to recruit new editors in the case of Wikipedia, coupled with a loss of current editors, would seem to suggest that the answer is yes (Crowston et al., this issue). So, what are the relevant concerns that participants of an open source ecosystem must be aware of when they decide to collaborate on an open source project? The seven articles in this issue (introduced below) each provide their own distinctive answer to this ques-

Editorial: Open Source Sustainability

Chris McPhee and Maha Shaikh

tion, and borrowing from them and my own research, I have framed an initial understanding of eight factors that influence open source sustainability.

Open development process
Most of the authors of this issue would argue that an open source development process is just as important as keeping the code open source. Many of the articles touch on this implicitly, but Ingram and Arikan show how, in the case of openEHR, an open process becomes necessary. If a universal electronic health record is ever to emerge and sustain itself over time, then keeping only the code alone does not allow the type and depth of co-creation and expertise-sharing necessary to build an accountable and legitimate system.

License promiscuity
Depending on the needs of a particular project (and these will vary from organization to organization), it has been argued that, if the license is more rather than less permissive, then the project has better chance of survival. Asay (in this issue) who has many years of experience of open source software adoption and management in commercial organizations, persuades us that an Apache-style license is conducive to sustainability in an open source ecosystem because it entirely frees the code and the creator. In contrast, the General Public License (GPL) demands greater reciprocity and, with companies becoming more experienced with open source co-creation and adoption, it has become a less attractive license. Given that companies are now playing a very relevant role in sustaining open source projects, perhaps greater consideration should be paid to the topic of license promiscuity .

Adaptable and innovative business models
Companies that moved into the open source arena very early on were typically motivated by strategic purposes rather than profit. Traditional business models did not apply, and it took organizations some years before open source could be exploited with clear and novel business models. As more companies and the public sector take deeper interest in open source, the business models we have to take advantage of this phenomenon and innovation need to be adapted accordingly – and fast. Sustainability is indeed more about changing rather than just change, as Curto-Millet (in this issue) explains.

Community
Most would agree that sustainability in open source means sustainability of the pool of developers that contribute to the code. But, how is a community kept healthy and vibrant, and more importantly, what is considered by ecosystem participants to be a sign of good health in a community? Crowston and colleagues (in this issue) explain that communities must manage recruitment with great care to keep themselves sustainable. However, we also note a change in attitude towards what amounts to a contribution in open source by a community. In the very early days, a contribution needed to be code-related, but with a growing diversity in ecosystem participants (both consumers and producers), there is a growing awareness of the value of other types of contributions, which can be as simple as just passing on the message about an open source project. This realization indicates a change in the nature of how and what we conceptualize as a community and as a valid contribution.

Open governance and accountable management
Different forms of governing an ecosystem, community, and organization lead to different outcomes. As Noori and Weiss (in this issue) argue, it is important for the long-term survival of an open source project and platform to adopt a governance style that changes and grows as the needs of the community change. This can be linked to Schweik's (in this issue) breakdown of a project lifecycle as stages of initiation and growth. How at each stage (and its variations) does governance become more governing-like and thus better able to manage change, growth, and then long-term sustainability over time? Flexibility may be the key to meeting these challenges, and as Curto-Millet (in this issue) argues, we therefore need to take a more process-oriented perspective.

Forking
Forking is often seen as a necessary evil in open source, but Nyman and Lindman show us another way to make sense of this process. They show that through governance and management at the levels of software, community and ecosystem, the right to fork can build greater strength and sustainability for the future.

Open source foundations
Open source foundations have had a presence for some while, but only recently has wide appreciation been given to their rather important role in keeping projects together by informing the community about various issues, offering legal protection, and providing governance through the development and implementation of rules and regulations. The number of foundations has grown, and Ingram and Arikan (in this issue) offer some possible causes for this change through their own example of openEHR and Opereffa. Sustainability in

open source projects does imply abiding by some form of regulations, standards, and codes of practice, all of which could slow the early stages of growth in a project. However, sustainability is not just about short-term thinking. In the long-term, Ingram and Arikan feel that some broader body of templates, archetypes, and rules will provide the infrastructure for a more sustainable open source project.

Ecosystem sustainability
Several of the articles in this issue move between sustainability at the levels of community, platform, and ecosystem. It can be argued that, because many projects are now built using a platform concept, to allow for an ecosystem to emerge around the code and participants, we need to be more focused on ecosystem sustainability rather than just sustainable communities.

Articles in this Issue

This issue contains seven articles relating to the theme of open source sustainability. The authors come from diverse backgrounds and geographical locations, including Canada, Finland, France, Spain, the United Kingdom, and the United States.

Linus Nyman and **Juho Lindman** from the Hanken School of Economics in Helsinki, Finland, argue that the ability to fork is a governance mechanism for ensuring sustainability in open source projects. Analysis at the levels of software, community, and ecosystem provide a more nuanced explanation of the motivations for forking, as well as the problems and benefits that can arise from it. Thus, the authors argue that forking need not be seen as negative behaviour; rather, it can be a way of building long-term sustainability.

Charles Schweik, Associate Professor at the University of Massachusetts, USA, discusses open source sustainability in relation to technological, community, and institutional attributes. Building on detailed survey data, Schweik adapts Ostrom’s (2005; tinyurl.com/aesc7vd) Institutional Analysis and Development (IAD) framework, where projects are seen in initiation or growth stages (and more subtle variations as well). Public sector organizations interested in open source, commercial organizations, and other open source project based communities will be particularly interested in Schweik's framework.

Kevin Crowston, from Syracuse University in the United States, **Nicolas Jullien**, from Telecom Bretagne in France, and **Felipe Ortega** from the University Rey Juan Carlos in Spain, have studied Wikipedia in various languages with a focus on one criteria of project sustainability: the recruitment and retention of participants. In their essay, two notions emerge strongly: i) the management of projects and how they are organized, their hierarchy, and their rules influence who is recruited as editors to projects, but also who joins and participates, and ii) the size and maturity of the project greatly impact sustainability and recruitment.

Nadia Noori, a graduate from the Technology Innovation Management (TIM) program at Carleton University in Ottawa, Canada, and **Michael Weiss**, an Associate Professor and TIM faculty member, move beyond a community perspective to explore platform sustainability. The sustainability of a platform depends on what form of governance is exercised over the platform, and the authors identify three types of governance model: tight-control, loose-control, and hybrid-control. Their article creates a link from the community (or individual, organizational perspective) to a platform and finally to the larger ecosystem.

David Ingram and **Sevket Seref Arikan** from University College London in the United Kingdom explain how the problem of building a universal electronic health record system could be (mostly) resolved by a reliance on not only open source software, but also on a very open source process of development. The open development process would need to be clearly designed and implemented so that others can imitate it and truly hope to co-create a universal health record system. In their discussion of openEHR, the Opereffa framework, and archetypes and templates, they make evident the need for openness, governance, and controlled management to build not simply a local system but a universal electronic health record system. Their in-depth case is based in the United Kingdom National Health Service (NHS), but their offer of a possible solution has universal applicability and appeal.

Daniel Curto-Millet, a doctoral student at the London School of Economics and Political Science in the United Kingdom encourages us to ontologically redefine sustainability. His study of openEHR and the Opereffa framework have shown him how sustainability is not a state that is stable (even in its desire for stability), but instead sustainability is a process where the multitude of actors, artefacts, archetypes, and so on, and are all in constant flux. He thus feels we need to conceptualize sustainability in a manner that allows us to make sense of it processually – in other words, as in "becoming" (Deleuze and Guattari, 1987; tinyurl.com/awujgdr). In order to be able to do this, he draws our attention to everyday negotiations, working outs, and engagements that openEHR and its larger ecosystem perform with and within to achieve a more detailed understanding of *sustaining* (and not sustainability).

Matt Asay, Vice President of Corporate Strategy at 10gen in the United States, discusses open source software ecosystem sustainability where the key issues according to his years of experience in the field are: i) community sustainability and ii) license permissiveness. He highlights the need to revaluate and redefine contribution in light of commercial interests in open source. This resonates strongly with my own findings and research in this area. OSS (open source software) has evolved into OSS 2.0 (Fitzgerald, 2006; tinyurl.com/dxwq3jx), and, whereas in the early days companies were considered parasitic by communities and developers, we now note a real shift. This shift in attitude is partly due to a changed understanding of contribution in open source – it no longer only implies a contribution of code (though this is still very relevant). It has taken on a more multifaceted role that is evident in practice, acceptance, and understanding. Contribution to open source can now be redefined to mean anything from code updates to use, interest, and generating a conversation on open source, activism, bug reports, training, education, and so on. Open source has truly grown up and matured. It has become more inclusive, malleable and perhaps in its more hybrid manifestations, even more interesting?

Maha Shaikh
Guest Editor

About the Editors

Chris McPhee is Editor-in-Chief of the *Technology Innovation Management Review*. Chris holds an MASc degree in Technology Innovation Management from Carleton University in Ottawa and BScH and MSc degrees in Biology from Queen's University in Kingston. He has over 15 years of management, design, and content-development experience in Canada and Scotland, primarily in the science, health, and education sectors. As an advisor and editor, he helps entrepreneurs, executives, and researchers develop and express their ideas.

Maha Shaikh is an Assistant Professor at Warwick University Business School. Prior to this, she was a Research Associate at the London School of Economics and Political Science (LSE). Other affiliations include the University of Limerick, where she worked on a number of projects including the OPAALS project with Professor Brian Fitzgerald. She has also worked with Professor Leslie Willcocks at the LSE, studying the relationship of open source to outsourcing, open innovation, and open business models. Dr Shaikh is a co-author of *Adopting Open Source Software: A Practical Guide*.

Citation: McPhee, C. and M. Shaikh. 2013. Editorial: Open Source Sustainability. *Technology Innovation Management Review*. January 2013: 3-6.

Code Forking, Governance, and Sustainability in Open Source Software

Linus Nyman and Juho Lindman

“*The ability to fork code – a central freedom of open source software – is what keeps communities vibrant and companies honest.*”

Glyn Moody
Technology writer and journalist

The right to fork open source code is at the core of open source licensing. All open source licenses grant the right to fork their code, that is to start a new development effort using an existing code as its base. Thus, code forking represents the single greatest tool available for guaranteeing sustainability in open source software. In addition to bolstering program sustainability, code forking directly affects the governance of open source initiatives. Forking, and even the mere possibility of forking code, affects the governance and sustainability of open source initiatives on three distinct levels: software, community, and ecosystem. On the software level, the right to fork makes planned obsolescence, versioning, vendor lock-in, end-of-support issues, and similar initiatives all but impossible to implement. On the community level, forking impacts both sustainability and governance through the power it grants the community to safeguard against unfavourable actions by corporations or project leaders. On the business-ecosystem level forking can serve as a catalyst for innovation while simultaneously promoting better quality software through natural selection. Thus, forking helps keep open source initiatives relevant and presents opportunities for the development and commercialization of current and abandoned programs.

Introduction

This article addresses the question of how the right to fork open source projects – to use the source code of an existing program to start a new, independent version – works as a governance mechanism to provide sustainability in open source software. The concept of sustainability is under debate, with numerous rubrics against which the sustainability of a product may be measured (e.g., Connelly, 2007: tinyurl.com/atjcgq3; Davison, 2001: tinyurl.com/aukl5ch; McManus, 1996: tinyurl.com/a5usfo3). Within the context of the current study, sustainability is defined as the possibility of an open source program to continue to serve the needs of its developers and users.

While code forking may lead to redundant independent efforts, it represents the single greatest tool available for guaranteeing sustainability in open source software. In this article, we examine code forking within open source initiatives and discuss the managerial implications of code forking. The article is structured as follows: first, we offer some background on code forking; second, we look at how code forking affects governance on the three levels mentioned; finally, we explain the relevance of these findings and their management implications.

Background

Code forking has often been viewed in a negative light. At the core of this negative view is the continued use of a restrictive, and perhaps outdated, definition of the term forking. Until recently, the term fork was mainly used to describe a situation in which a developer community had split into competing camps, each continuing work on their own, incompatible version of the software (see, for example, Raymond, 1999: tinyurl.com/3ald3; Fogel, 2006: tinyurl.com/3dx2py). Hence, the negative tone found in discussions of forking has been related to concerns regarding the hindered progress, wasted resources, and potential demise of one or both of the projects. In recent years, the term forking has come to be used in a much broader context, encompassing all cases in which one takes an existing code base and implements it in a separate project (see, for instance, GitHub: tinyurl.com/7uc94sk). In the context of this study, we adhere to this broader definition of forking.

While there are many reasons why projects are forked, the most common reason is the desire to modify the original program to better address a specific need (Nyman and Mikkonen, 2011; tinyurl.com/arntyur). Forks may also be planned, temporary divergences intended to test new ideas and features, with the intention of later integrating effective improvements back into the original (Nyman and Mikkonen, 2011: tinyurl.com/arntyur; see also GitHub: tinyurl.com/7uc94sk). The right to fork code is built into the very definition of what it means to be an open source program. The third criteria of the Open Source Initiative's (OSI; opensource.org/osd.html) definition of open source states that the license "must allow modifications and derived works." Similarly, the Free Software Foundation's Free Software Definition (FSD; gnu.org/philosophy/free-sw.html) states that users have the freedom to "run, copy, distribute, study, change and improve the software." All spinoff initiatives can be considered forks as they are "modified or derived" (OSI) or "copied, changed and improved". The possibility of forking any project affects the governance and sustainability of all open source programs.

Software is editable, interactive, reprogrammable, distributed, and open (Kallinikos et al., 2010; tinyurl.com/4zn6cun). These characteristics dictate that software is prone to being changed, repaired, and updated rather than remaining fixed from the early stages of the design process. The openness combined with the granular composition of the software offer new ways of governance (Benkler, 2006; tinyurl.com/6ftot3). This governance is not tied to over-appropriating a natural resource (Ostrom, 1991; tinyurl.com/b8rc2pu), but rather related to ways in which a group of developers, following institutional rules, collectively produce a public good (Schweik et al., 2010; tinyurl.com/aqxy2jp).

Three Levels of Governance

1. Software level
The nature of the industry dictates that programs cannot maintain a stable steady state for an extended period of time. They must continue to evolve in order to remain useful and relevant. Without continual adaptation, a program will progressively become less satisfactory (Lehman, 1980; tinyurl.com/b2mpkw3). Conversely, truly successful software is able to adapt and even outlive the hardware for which it was originally written (Brooks, 1975; tinyurl.com/awg3rrw). Therefore, the ability to change and evolve is a key component of software sustainability. Although stagnation may be a precursor to obsolescence, obsolescence need not creep into a project over time; it is often a design feature.

Popularized in the 1950s by American industrial designer Brooks Stevens (The Economist, 2009; tinyurl.com/ahws66g), the concept of planned obsolescence stands in stark contrast to the concept of sustainability. Stevens defined planned obsolescence as the act of instilling in the buyer "the desire to own something a little newer, a little better, a little sooner than is necessary" (Brooks Stevens' biography; tinyurl.com/bbs8a3c). Considered "an engine of technological progress" by some (Fishman et al., 1993; tinyurl.com/bye2n5r), yet increasingly problematized in the business ethics literature (Guiltinan, 2009; tinyurl.com/alr2c92), planned obsolescence is part of every consumer's life. Although contemporary software development and distribution have characteristics that differ substantially from the industrial products of the 1950s, the revenue models of companies in the software marketplace often welcome elements such as system versioning, to encourage repurchases of a newer version of the same system, or vendor lock-ins that limit the customer choice to certain providers of system or product (for a further review, see Combs, 2000; tinyurl.com/aq2wl7h). Newer versions of programs may introduce compatibility problems with earlier operating systems or programs (e.g., lack of backwards compatibility in Internet Explorer, Microsoft Office, or OS X's OpenStep APIs). Some programs also introduce new file formats, which can cause compatibility issues with earlier versions of the program (e.g., docx vs. doc). Furthermore, end-of-life announcements and concerns over end-of-support deadlines may encourage users to upgrade, regardless of the real need to do so.

The right to fork code makes implementing such elements impracticable in open source. The right to improve a program, the right to combine many programs, and the right to make a program compatible with other programs and versions are all fundamental rights that are built into the very definition of open source. Research has shown these rights are often exercised (Fitzgerald, 2006; tinyurl.com/al995aj). The result of this constant collaborative improvement in open source systems is that any program with the support of the open source community can enjoy assured relevance rather than planned obsolescence. Furthermore, with renewed community interest, programs that have decayed and fallen into disuse can be revived and updated by forking the code from the original program. In fact, this is a fairly common practice: of the almost 400 forks studied by Nyman and Mikkonen (2011; tinyurl.com/arntyur), 7% involved the reviving of an abandoned project. As long as there is sufficient community interest in a project, forking can allow for constant improvement in software functionality.

2. Community level
The possibility to fork is central to the governance of any open source community. The shared ownership of open source projects allows anyone to fork a project at any time. Therefore, no one person or group has a "magical hold" over the project (Fogel, 2006; tinyurl.com/ahbh8nt). Since a fork involving a split of the community can hurt overall productivity, Fogel notes that the potential to fork a program is "the indispensable ingredient that binds developers together".

One of the concerns among open source communities is what Lerner and Tirole (2002; tinyurl.com/bfmaxl4) call the hijacking of the code. Hijacking occurs when a commercial vendor attempts to privatize a project's source code. The 2008 acquisition of MySQL (mysql.com), an open source relational database management system, by Sun Microsystems and subsequent acquisition of Sun by Oracle is an example of a case involving community concern over potential hijacking. It had been argued that such a series of acquisitions would lead to the collapse of both MySQL and the open source movement at large (Foremski, 2006; tinyurl.com/yesjhw7). Responding to such claims, Moody (2009; tinyurl.com/cbrq7g) noted that, while open source companies can be bought, open source communities cannot. Forking provides the community that supports an open source project with a way to spin off their own version of the project in case of such an acquisition. Indeed, this is what happened in the case of MYSQL. The original MySQL developer, Michael ("Monty") Widenius, forked the MySQL code and started a new version under a different name, MariaDB, due to concerns regarding the governance and future openness of the MySQL code (for details, see Widenius' blog [February 5, 2009: tinyurl.com/btr9bm6 and December 12, 2009: tinyurl.com/ba58vpp] and press release [tinyurl.com/auvaxbn]).

Similarly, in 2010, community concerns regarding governance led to a forking of the OpenOffice (OO; openoffice.org) project. The Document Foundation, which included a team of long-term contributors to OO, forked the OO code to begin LibreOffice (libreoffice.org). The spinoff project emphasized the importance of a "transparent, collaborative, and inclusive" government (The Document Foundation; tinyurl.com/bzmw5p2). A recent analysis of the LibreOffice project indicates that this fork has resulted in a sustainable community with no signs of stagnation (Gamalielsson and Lundell, 2012; tinyurl.com/a9ev4hu). Given that forking ensures that any project can continue as long as there is sufficient community interest, we have previously described forking as the "invisible hand of sustainability" in open source software (Nyman et al., 2011; tinyurl.com/b8bzorg).

Commonly, forking occurs due to a community's desire to create different functionality or focus the project in a new direction. Such forks are based on a difference in software requirements or focus, rather than a distrust of the project leaders. When they address disparate community needs, different versions can prosper.

In a traditional company, it is the management, headed by the CEO and board of directors, that controls the company and provides the impetus for continued development. While the vision of the leadership is similarly integral to the eventual success of any open source project, their continued control is more fragile and hinges upon their relationship with and responses to the community. Forking cannot be prevented by business models or governance systems. The key lies in appropriate resource allocation and careful community management. Managers must strike a delicate balance between providing a driving force while appeasing and unifying the community. (For an overview of open source governance models, see OSS Watch [tinyurl.com/bjqpnkn]; for discussion on building technical communities, see Skerrett, 2008: [timreview.ca/article/160]; for discussion on open source community management, see Byron, 2009: [timreview.ca/article/258].)

3. Business-ecosystem level
Within the dynamic world of open source software, natural selection acts as a culling force, constantly choosing only the fittest code to survive (Torvalds, 2001; tinyurl.com/aaxqux7). However, the right to fork means that any company can duplicate any competitor's open source software distributions; thus, competitive advantage cannot depend on the quality of the code alone. However, it is worth stressing that possibility does not equal success. The right to fork a commercially successful program with the intention of competing for the same customer base still leaves the would-be competitor with issues regarding trademarks, brand value and recognition, as well as the existing developer and user base of the original program. Even though forking allows companies to compete with identical open source software, it is nevertheless cooperation that is con-

sidered to be the key to corporate success (Skerrett, 2011: timreview.ca/article/409; Muegge, 2011: timreview.ca/article/495).

Open source software is free, but it is also increasingly developed and supported for commercial gains (Wheeler, 2009: timreview.ca/article/229). While the right to fork may seem to make for a harsh business environment, open source companies can and do thrive. With its billion-dollar revenue (tinyurl.com/b7py36u), Red Hat is one such example. While their revenue primarily comes from subscriptions and services related to their software (see Suehle's [2012; timreview.ca/article/513] TIM Review Q&A for a more in-depth look at the secret of Red Hat's success), Red Hat's programs themselves are largely based on forks of programs by other developers. This phenomenon of combining forked programs is not unique to Red Hat: the hundreds of different Linux distributions (tinyurl.com/85r9o) are all made possible by the forking of existing products and repackaging them as a new release.

Forking lays the building blocks for innovators to introduce new functionalities into the market, and the plethora of online forges have hundreds of thousands of programs available for forking and reuse in any new, creative way the user can imagine, allowing for the rapid adaptation to the needs of end users. Hence, the practice of forking allows for the development of a robust, responsive software ecosystem that is able to meet an abundance of demands (Nyman et al., 2012; tinyurl.com/acg3fp2).

The old adage, "one man's trash is another man's treasure" is particularly salient in open source software development. Soon after Nokia's abandonment of the MeeGo project in 2011 (press release: tinyurl.com/ad5lh6b; MeeGo summary: tinyurl.com/9u4xrno), the Finnish company Jolla announced that it would create a business around its revival, made possible by forking the original code (press release: tinyurl.com/7bzbo9h). On July 16, 2012, Jolla announced a contract with D. Phone, one of the largest cell phone retailers in China, and on November 21 they launched Sailfish OS (tinyurl.com/a4yot8h). However, one does not need to be an open source business to benefit from the right to fork. Forking can also aid companies who choose to use an existing program, or develop it for personal use. The requirement in open source to share one's source code is linked with distribution, not modification, which means that one can fork a program and modify it for in-house use without having to supply the code to others. However, a working knowledge of licenses as well as license compatibility (when combining programs) is crucial before undertaking such an endeavour (for a discussion of licenses, see St. Laurent [2004; tinyurl.com/befxwvc], Välimäki [2005; tinyurl.com/ahljzwu], or Meeker [2008; tinyurl.com/am93qol] for a discussion of architectural design practices in the combining of licenses, see Hammouda and colleagues [2010; tinyurl.com/bfp82mw].

A summary of the ways in which forking can affect governance and help ensure sustainability is provided in Table 1.

Managerial Implications

Managers should consider the following implications of code forking:

• An abandoned project can become a business opportunity.

• Neither business models nor governance systems can completely prevent forking. Thus, developer and community satisfaction is of key importance.

• A strong, vibrant community is a key issue to consider when implementing an open source program. When acquiring systems, the potential of forking in open source software – in particular when coupled with a strong community – provides opportunities to avoid versioning and vendor lock-in to one provider of a product or system. However, while community is important, it is not the only factor to consider. For more on evaluating and selecting open source software for corporate use, see the May 2008 issue of *TIM Review*, including topical articles by Golden (2008; timreview.ca/article/145), von Rotz (2008; timreview.ca/article/147), and Semeteys (2008; timreview.ca/article/146).

• There are thousands of open source programs already in existence, which can be forked. If a need for software arises and open source is an option, begin by analyzing what already exists on code repositories such as SourceForge (sourceforge.net) and GitHub (github.com). Keep in mind that it is distribution, not modification, that obligates the sharing of the source code. Be sure to read up on licenses first!

Conclusion

Forking sits at the intersection of several different open source topics, such as software development, governance, and company participation in communities and business ecosystems. In the interest of clarity, we have simplified the categorization of the multifaceted concept of forking. In actuality, there is overlap among the categories: a strong community offers better insurance of sustainability of the software level, while better software can more easily attract a bigger community. Both a poorly handled community and an abandoned project can spawn a business ecosystem competitor.

The right to fork code is intrinsic to open source software and is guaranteed by all open source licenses. This right to fork has a significant effect on governance and helps ensure the sustainability of open source software. We have analyzed the effect of forking on three different levels: the software level, the community level, and the ecosystem level. On a software level, code forking serves as a governance mechanism for sustainability by offering a way to overcome planned obsolescence and decay, as well as versioning, lock-in, and related concerns. On a community level, code forking ensures sustainability by providing the community with an escape hatch: the right to start a new version of the program. Finally, on an ecosystem level, forking serves as a core component of natural selection and as a catalyst for innovation. Online forges offer a plethora of publically available programs that can serve as the building blocks of a new creation. Current projects can be forked, abandoned projects can be revived and commercialized, or programs can be combined in novel ways to better meet the needs of both the developers and end users. It is the right to fork that moulds the governance of open source projects and provides the dynamic vigour found in open source computing today.

Table 1. Forking and its effect on governance

Level	How Forking Provides Sustainability	Examples
Software	The right to fork protects against planned obsolescence, versioning, and vendor lock-in	Microsoft Word vs. LibreOffice
	Disuse due to decay can be countered by forking and updating	Fairly common open source practice (for examples, see Nyman [2011; tinyurl.com/arntyur])
Community	Prevents hijacking and other unfavorable actions by project leaders or owners through giving developers the option to continue their own version of the program	MariaDB forked from MySQL, LibreOffice forked from OpenOffice
Ecosystem	Increases innovative potential by allowing for the combination and modification of open source projects	Plethora of different Linux distributions
	Abandoned (or badly handled) projects can be revived, creating new business opportunities	(Abandoned) MeeGo forked to create Sailfish

Code Forking, Governance, and Sustainability in Open Source Software

Linus Nyman and Juho Lindman

About the Authors

Linus Nyman is a doctoral student at the Hanken School of Economics in Helsinki, Finland, where he studies code forking in open source software. When not researching, he can sometimes be found lecturing on corporate strategy or open source software. Other areas of interest include freemium business models and MMORPGs (online gaming). Linus has a Master's degree in economics from the Hanken School of Economics.

Juho Lindman is an Assistant Professor of Information Systems Science at the Hanken School of Economics in Helsinki, Finland. Juho defended his doctoral dissertation focusing on open source software development organization at the Aalto University School of Economics in Helsinki. In the field of information systems, his current research is focused in the areas of open source software development, open data, and organizational change.

Citation: Nyman, L. and J. Lindman. 2013. Code Forking, Governance, and Sustainability in Open Source Software. *Technology Innovation Management Review.* January 2013: 7-12.

Sustainability in Open Source Software Commons: Lessons Learned from an Empirical Study of SourceForge Projects

Charles M. Schweik

“*The real free-rider problems in open-source software are more a function of friction costs in submitting patches than anything else. A potential contributor with little stake in the cultural reputation game... may, in the absence of money compensation, think "It's not worth submitting this fix because I'll have to clean up the patch, write a ChangeLog entry, and sign the FSF assignment papers...". It's for this reason that the number of contributors (and, at second order, the success of) projects is strongly and inversely correlated with the number of hoops each project makes a contributing user go through.*”

Eric Raymond
Computer programmer, author, and open source advocate
in *The Cathedral and the Bazaar*

In this article, we summarize a five-year US National Science Foundation funded study designed to investigate the factors that lead some open source projects to ongoing collaborative success while many others become abandoned. Our primary interest was to conduct a study that was closely representative of the population of open source software projects in the world, rather than focus on the more-often studied, high-profile successful cases. After building a large database of projects (n=174,333) and implementing a major survey of open source developers (n=1403), we were able to conduct statistical analyses to investigate over forty theoretically-based testable hypotheses. Our data firmly support what we call the conventional theory of open source software, showing that projects start small, and, in successful cases, grow slightly larger in terms of team size. We describe the “virtuous circle” supporting conventional wisdom of open source collaboration that comes out of this analysis, and we discuss two other interesting findings related to developer motivations and how team members find each other. Each of these findings is related to the sustainability of these projects.

Introduction

This special issue of the *TIM Review* is devoted to questions surrounding the idea of “sustainability” in relation to open source software. The call for papers asked authors to connect some of Elinor Ostrom's work (1990: tinyurl.com/b3neybk; 2005: tinyurl.com/aesc7vd; 2010: tinyurl.com/aasko9e) related to sustainability, collective action, and the commons and apply it to open source. Over the last seven years, my research team and I have been doing just that. In this article, I summarize how we connected to Ostrom's approach to studying the commons and report some of the important findings related to questions of sustainability in open source software commons. The article focuses on the practical implications of the research findings.

Sustainability in Open Source Software Commons

Charles M. Schweik

Our Research Perspective

The overarching research question driving our research is: *What factors lead some open source software commons to success and others to abandonment?*

At the heart of this question is sustainability of open source software, from a *collaboration* perspective. Why do some programmers stay with a project while others leave? Here we focus not only on open source volunteer programmers – a central theme in many previous studies of open source – but paid programmers as well. Further, a central goal of our work was to investigate not simply high-profile, large-scale success stories (e.g., Linux, Apache Web Server), as was the case with much of the early research on open source, but to get a better handle on the unknown population of open source software projects, which at the time we started our work (~2005) was certainly well over 100,000 in number.

To begin our research, we built upon Elinor Ostrom and colleague's Institutional Analysis and Development (IAD) framework (Ostrom, 2005: tinyurl.com/aesc7vd; Figure 1). In this framework, as it applies to open source software commons, a central unit of analysis is the individual open source developer (diamond in Figure 1) who we assume is a boundedly rational actor and who periodically reflects on whether or not they should continue contributing to the project. This logic, at any point in time, is based in part on three groups of variables or influential factors that might contribute influence the developer's decision, depicted on the left hand side of Figure 1: i) Technological, ii) Community, and iii) Institutional attributes of the open source software project. In Schweik and English (2012; tinyurl.com/ap6cxuw), we review a significant amount of theoretical and empirical literature in an effort to identify important factors that are thought to influence other types of commons (such as natural resource commons) or are

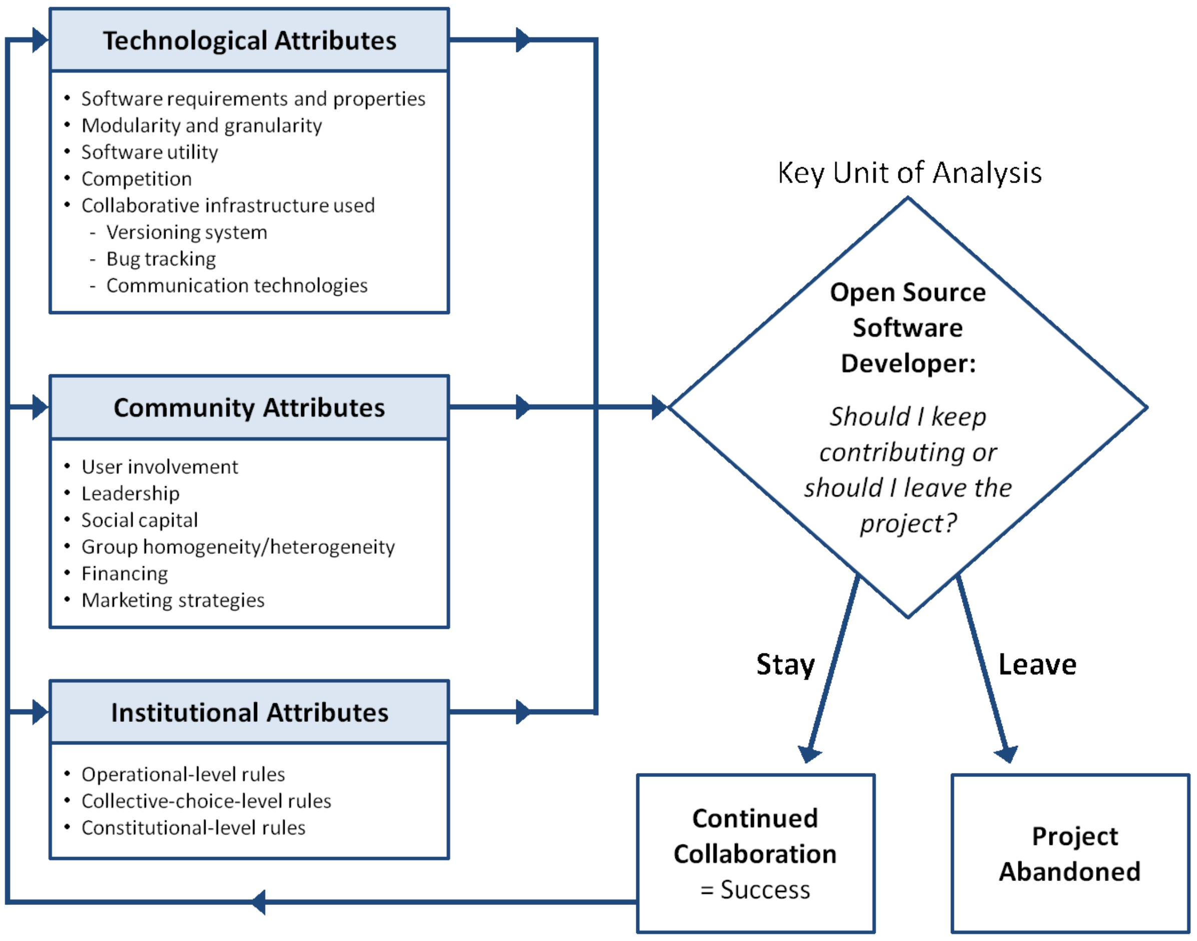

Technological Attributes
Software requirements and properties
Modularity and granularity
Software utility
Competition
Collaborative infrastructure used
Versioning system
Bug tracking
Communication technologies
Community Attributes
User involvement
Leadership
Social capital
Group homogeneity/heterogeneity
Financing
Marketing strategies
Institutional Attributes
Operational-level rules
Collective-choice-level rules
Constitutional-level rules
Key Unit of Analysis
Open Source Software Developer:
Should I keep contributing or should I leave the project?
Stay
Leave
Continued Collaboration = Success
Project Abandoned

Figure 1. A simplified institutional analysis and development framework to support analysis of sustainability in open source software commons. Adapted from Ostrom (2005; tinyurl.com/aesc7vd) and Schweik & English (2012; tinyurl.com/ap6cxuw).

thought to influence the sustainability of software projects. This included literature specifically on open source, but also software engineering, virtual teamwork, and environmental commons or common property (e.g., forests, fisheries, irrigation systems). The three groups of attribute on the left side of Figure 1 list some of the factors – but not all – we identified through this work. To give the reader an idea of these three attribute groupings, let us consider an example of each.

A Technological Attribute thought to influence a developer's decision to stay with a project or leave might be related to "task granularity" as Yochai Benkler (2006; tinyurl.com/6ftot3) puts it; if the development task is too large or “coarse grained”, the developer might decide it requires too much effort for the volunteer (or paid) time he or she can allocate to it and might decide to leave the project.

A Community Attribute thought to influence a developer's decision to stay or leave might be the attributes of the leader(s) of the project. Leadership is a complicated variable or set of variables, but one aspect of it relates to the idea of leading by example; leaders motivate others on the team to do work by contributing significant work themselves.

An Institutional Attribute thought to influence a developer's willingness to stay with a project or leave might be the level of formality required to participate on the project. A famous proponent of open source, Eric Raymond (2001; tinyurl.com/d546xlv) described formalized rules for collective action in open source as “friction” that creates negative incentives for contribution (see the introductory quote above). Space limits us to describe all the variables we investigated in this study, but the topics listed in the three boxes on the left side of Figure 1 will give the reader a sense of the kinds of variables we investigated. Ultimately, we identified over 40 variables, most of which led to testable hypotheses where *a priori* expectations on their influence were known. However, in some cases, we had no idea what relationship would be found, and no previous theory or empirical work to suggest an expected relationship with our dependent variable, success or abandonment of open source software projects.

The reader should note that Figure 1 represents a dynamic system that changes over time. As long as a project stays operational, there is feedback threading back to the three sets of attributes to the left in Figure 1, and periodically, these attributes might change in some dimension. These changes then have an effect or may influence the developer's feelings about the project and their periodic reflections on whether to stay or leave, and the cycle continues.

Methods

To begin our empirical work, we first searched for a dataset on open source software projects that was already collected, rather than having to build one from scratch. Fortunately, a group called FLOSSMole (flossmole.org) based out of Syracuse University had been actively scraping the dominant open source project hosting site SourceForge (sourceforge.net) and building a database on these projects for other researchers to use (Howison et al., 2006; tinyurl.com/abounnq). Their database contained metadata about these projects, most related to Technological or Community-related attributes, but with at least one Institutional variable (license used). Our initial SourceForge database, gathered in the summer of 2006, contained 107,747 projects. In 2009, we collected a second time-slice from a different repository called the SourceForge.net Research Data Archive (tinyurl.com/ard7v9z), which is housed at the University of Notre Dame. This second dataset, representing SourceForge projects in 2009, contained 174,333 projects.

Our next step was to formulate a measure of success and abandonment for open source software projects. This was a challenging endeavour, which took us over a year and a half to complete. We first identified two different longitudinal stages that open source projects go through: i) an Initiation Stage and ii) a Growth Stage. The Initiation Stage describes the period of time from project start to the first public release of software. On the SourceForge hosting site, it is easy to find new projects that have yet to make code available to the public but are being actively worked on. We use the Growth Stage to describe the period after a project's first public release of code. One could conceptualize a “termination” or “abandonment stage” as well, but in our conceptualization, that particular event can occur in either the Initiation Stage (pre-first release) or in the Growth Stage (post-first release).

With these two stages defined, we then set out to carefully define, both theoretically and empirically, a method to measure whether a project is successful or abandoned in these two stages. We identified six categories of success and abandonment: Success in Initiation (SI); Abandonment in Initiation (AI); Success in

Growth (SG); Abandonment in Growth (AG); Indeterminate in Initiation (II); and Indeterminate in Growth (IG). Details of this initial phase of our research can be found in English and Schweik (2007; tinyurl.com/bd29rnu). Our classification system was later replicated independently by Wiggins and Crowston (2010; tinyurl.com/a33k9fn). Table 1 presents our definitions and results for the 2006 SourceForge dataset; for the results from our 2009 SourceForge data, please see Schweik and English (2012; tinyurl.com/ap6cxuw).

These datasets provided an excellent start, but our mapping of SourceForge projects to the identified theoretical variables (Figure 1) led to the conclusion that many of the community and institutional variables we wanted to investigate were not captured in these datasets. Consequently, in 2009, we implemented a complementary online survey for SourceForge developers to capture these missing variables. The challenge was that, if we contacted a random sample of SourceForge project administrators, we expected that we would get significant bias toward successful collaborations that were active. To ensure enough responses from abandoned projects, we needed to sample a much larger number of SourceForge projects. In the summer of 2009, we stratified our 2009 dataset using our success/abandonment classification and randomly selected 50,000 projects to survey. With the help of the SourceForge organization, we emailed a survey to the SourceForge project administrators for each of these projects. The result: 1403 surveys returned.

Table 1. Success and abandonment categories for open source software projects in the 2006 SourceForge database

Class	**Definition**	**Number of Projects***
SI: Success in Initiation	Developers have produced a first release.	–
AI: Abandonment in Initiation	No first release produced, and the project appears to be abandoned.	37,320 (35%)
SG: Success in Growth	Project has achieved 3 meaningful releases of the software and the software is deemed useful for at least a few users.	15,782 (15%)
AG: Abandoned in Growth	Project appears to be abandoned before producing 3 releases of a useful product, or has produced 3 or more releases in less than 6 months and is abandoned.	30,592 (28%)
II: Indeterminate in Initiation	Project has yet to reveal a first public release but shows significant developer activity.	13,342 (12%)
IG: Indeterminate in Growth	Project has not yet produced three releases but shows development activity, or has produced three releases or more in less than six months and shows development activity.	10,711 (10%)
Total projects		**107,747**

* Successful Initiation (SI) numbers are not listed because these successes are Growth-Stage projects; including the SI category would double-count projects.

With the online survey conducted, we were able to create a database of these 1403 projects and combine it with the SourceForge metadata from the 2009 Notre Dame dataset. We had a complete dataset capturing both our dependent variable of success and abandonment for all Initiation Stage and Growth Stage projects, as well as measures for our independent variables related to Technological, Community, or Institutional attributes. The dataset captured more than 40 independent variables, a small sample of which are listed in Figure 1.

We used three statistical techniques to analyze the data. To investigate relationships of individual variables, we used contingency tables to investigate the differences in distribution for the projects as they relate to success and abandonment. We also used two different multivariate analytic techniques: i) classification and regression trees and ii) logistic regression. Full explanations of these techniques, as well as summary tables and results are available in Schweik and English (2012; tinyurl.com/ap6cxuw).

Selected Findings

Our analysis focused on over 40 variables thought to potentially influence whether open source projects maintained collaboration or whether they became abandoned. In this section, we will focus on some of our more general or most interesting findings, with a focus on practical insights.

First, we have empirical support for the conventional thinking of how open source software projects operate. The vast majority of open source projects do not have large teams, but rather have very small teams of one to three developers. Based on careful analysis of both Initiation Stage and Growth Stage data, we found that the majority of these projects tend to start with a very small development team of one to two developers and very little or no user community. Then, as work progresses and after a first release is made, a user community is established and grows over time. The founding developer(s) lead through doing, and through the development of a product that they often need (supporting von Hippel's [2005; tinyurl.com/57xp5x] idea of “user-driven need”), build something usable and, at the same time, begin to generate a user community. Through the regular open source communication channels (e.g., IRB sessions, email lists, websites, and bug tracking systems), they build social capital between themselves and their user base, and gradually grow their user base, and a virtuous cycle begins. More progress is made on the code base, leading to (potentially) a larger user base, and leading to (perhaps) an added developer. But, our study may be some of the first empirical results that actually capture this conventional thinking of how open source collaboration operates.

We also discovered that the successful Growth Stage projects tend to gain one developer compared to abandoned ones and, to our surprise, we found that over 58% of our successful projects gained a developer from *another continent*. This last point is quite striking, for we found that in many cases these new developers have never met face-to-face in person with other developers on the project but know and trust each other as a result of almost strictly Internet-based interaction. These findings align with what we have heard from open source developers we have interviewed.

Based on what we have found, related to the idea of open source project sustainability, the advice we have for leaders of projects in the Initiation Stage is:

1. Be ready to put in the hours. Work hard toward the creation of the first software release.

2. Demonstrate and signal good leadership by administering your project well and clearly articulating your vision and goals through project communication channels (e.g., website, bug tracking system). Create and maintain good documentation for potential new developers and for your user community through these channels.

3. Advertise and market your project and communicate the plans and goals, especially if you seek new developers to move the project forward over the longer term.

4. Realize that, in our data, successful projects are found in either GPL-compatible or non-GPL-compatible free/libre open source licenses.

5. When starting a project, consider its potential to be useful to a substantial number of users. The more potential users you have, the higher the likelihood that one or more of those users will have relevant skills and interests to consider joining and contributing to your project down the road.

Sustainability in Open Source Software Commons

Charles M. Schweik

Our advice for leaders of projects in the Growth Stage (post-first release) includes:

1. Focus on the idea of creating and maintaining the "virtuous circle", where good initial products attract users, which then potentially attract a new developer, which leads to more improvements. Our research clearly shows that successful projects have a potentially significant user community and that this user community drives project continuity.

2. Make sure that there are tasks of various sizes or effort demands that people can contribute to. Successful Growth Stage projects tend to have tasks for people to work on that fit into their available schedules. We remind readers of the concept of task granularity by Benkler (2006; tinyurl.com/6ftot3), mentioned earlier.

3. Surprisingly, our data suggests that competition seems to favour success rather than hinder it. In other words, do not give up if some competition appears on the horizon.

4. Financing helps.

5. To the extent possible, keep rules governing project collaboration and project governance lean and informal. To a large measure, the operational rules that do exist in open source software projects are often embedded in the version control systems that support the projects (e.g., CVS, Subversion), or are simple group-established social norms. We found that the vast majority of the projects we studied had very little *formalized* governance and operated under "Benevolent Dictator" type governance structures. In other words, they tend to support our opening quote by Eric Raymond (2001; tinyurl.com/d546xlv). Our sense from our study that simple, agreed-upon norms tend to drive these projects is in part because the vast majority of the projects we studied are very small teams that need very little in terms of formal coordination. However, we did have evidence that, as teams increase in size, project governance moves toward more formalized systems. Our evidence is fairly limited because, in our dataset, a very small proportion of the projects studied had large teams with 10 or more developers. But, this suggests that, if a project team grows, the team should not hesitate to move toward more formalized systems if required.

Our data analysis also led to some theoretical findings related to sustainability of open source software projects. The two most interesting of these findings are described below.

1. Developer motivations

Regarding questions of why developers participate in open source software projects, our results support much of the existing empirical work done earlier. Across both abandoned and successful projects, a primary motivator for participation was von Hippel's (2005; tinyurl.com/57xp5x) user-centric need. Developers participate because they themselves are users of the software or because the organization they work for depends on it. Other developers participate because they learn from the process of reading others' code and then developing new functions for the product. Others participate as a kind of "serious leisure" where they use their programming skills that they use for their employment and apply it to something outside of their work domain for their enjoyment. The one motivation that past research has suggested is important – that we found was not important – is the idea of signaling programming skills to others, often in an effort to possibly find eventual employment. In our survey data, this was not reported as an important factor and, in our view, it is because the vast majority of the teams are quite small (i.e., 1–3 people). But, perhaps the most interesting and new finding regarding motivations for participation in our research is our finding that projects with developers who have *multiple motivations* driving their participation will be more successful than projects with developers with only one motivation. In other words, open source projects will be more sustainable if individual members on the team have multiple reasons (e.g., "I learn and am paid to participate", or "I contribute because I am contributing to a public good and because I enjoy working on the project") driving their interests to contribute.

2. Sourceforge and Google as intellectual matchmakers

Some of our most careful work in this study revealed that successful open source software projects gain a developer and that quite often this new developer is not physically co-located with the developer(s) who founded the project, but rather, are geographically distant, and often on another continent. This provides some strong evidence suggesting that well-known websites for open source software, such as SourceForge, coupled with web search engines such as Google, create an intellectual matchmaker of sorts through "power-law typology" (Karpf, 2010; tinyurl.com/b6cxpzb). These power-law hubs are locations on the Internet that provide value to their users in part because of the network effects created because they have large crowds of similar users. Regardless of where a programmer lives in the world, people can find software projects that are related to this need and, over time, build social capital with the developers and eventually join the team if they speak the same language and demonstrate the desire and the skills needed to collaborate.

Conclusion

In this article, we described a five-year US National Science Foundation research study on the factors that lead some open source projects to ongoing collaboration and others to abandonment. To summarize, we find strong empirical support for the conventional wisdom of how open source software projects are sustained (see the virtuous circle discussion above) and report two of the most interesting findings of the study: i) that projects will be more sustainable if developers have multiple incentives driving their participation; and ii) successful projects gain a developer and this is likely driven through the intellectual match-making created by search engines such as Google coupled with power-law hubs such as SourceForge. For more detail on the research reported here, see Schweik and English (2012; tinyurl.com/ap6cxuw).

Acknowledgements

This research was supported by the U.S. National Science Foundation under grant number 0447623. The findings, recommendations, and opinions expressed here, however, are my own and do not necessarily reflect the views of the NSF. I am also very grateful to my research team: Robert English, Sandy Haire, Meng-Shiou Shieh, and others, without whom this work would not have been accomplished.

About the Author

Charles M. Schweik (Charlie) is an Associate Professor with a joint appointment between the Department of Environmental Conservation (eco.umass.edu) and the Center for Public Policy and Administration (masspolicy.org) at the University of Massachusetts Amherst. He is Associate Director of the National Center for Digital Government (ncdg.org) and the founding member of a new "Workshop on the Study of Knowledge Commons" on campus. His research focuses on environmental management and policy, public-sector information technology, and the intersection of those domains.

Citation: Schweik, C. M. 2013. Sustainability in Open Source Software Commons: Lessons Learned from an Empirical Study of SourceForge Projects. *Technology Innovation Management Review.* January 2013: 13-19.

Sustainability of Open Collaborative Communities: Analyzing Recruitment Efficiency

Kevin Crowston, Nicolas Jullien, and Felipe Ortega

“*In the great mass of our people there are plenty individuals of intelligence from among whom leadership can be recruited.*”

Herbert Hoover
31st President of the United States

Extensive research has been conducted over the past years to improve our understanding of sustainability conditions for large-scale collaborative projects, especially from an economic and governance perspective. However, the influence of recruitment and retention of participants in these projects has received comparatively less attention from researchers. Nevertheless, these concerns are significant for practitioners, especially regarding the apparently decreasing ability of the main open online projects to attract and retain new contributors. A possible explanation for this decrease is that those projects have simply reached a mature state of development. Marwell and Oliver (1993; tinyurl.com/bapafxc) and Oliver, Marwell, and Teixeira (1985; tinyurl.com/bal2y5y) note that, at the initial stage in collective projects, participants are few and efforts are costly; in the diffusion phase, the number of participants grows, as their efforts are rewarding; and in the mature phase, some inefficiency may appear as the number of contributors is greater than required for the work.

In this article, we examine this possibility. We use original data from 36 Wikipedias in different languages to compare their efficiency in recruiting participants. We chose Wikipedia because the different language projects are at different states of development, but are quite comparable on the other aspects, providing a test of the impact of development on efficiency. Results confirm that most of the largest Wikipedias seem to be characterized by a reduced return to scale. As a result, we can draw interesting conclusions that can be useful for practitioners, facilitators, and managers of collaborative projects in order to identify key factors potentially influencing the adequate development of their communities over the medium-to-long term.

Introduction

Mobilizing hundreds of contributors, as in the case of Linux, to thousands of contributors, as in the case of Wikipedia, open online communities (tinyurl.com/bbhkeuc) are viewed as a central point for innovative generation of new knowledge (Chesbrough, 2003: tinyurl.com/ce6bsy8; Mahr and Lievens, 2012: tinyurl.com/akozt3b). Open source initiatives are numerous and they span various industries (Balka et al., 2009; tinyurl.com/yaxxs3a). The success of such projects provides a new perspective on a fundamental socio-economic question in today's society: the sustainability of participation in collective action. The Olson paradox (Olson, 1965; tinyurl.com/bdj4usa) suggests that large groups are less able than small ones to promote their common interest because individual incentives to contribute diminish with group size. However, many communities, of all sizes and in various contexts, have shown their ability to develop selective incentives and institutions, making them able to develop and protect their “commons” (Ostrom, 1990: tinyurl.com/b3neybk; Hess and Ostrom, 2006: tinyurl.com/bamczvb). The question then is if these communities will be able to sustain their activities over the long term.

Recruiting and retaining new members is a recurrent issue for open communities, a topic already stressed and studied by von Krogh, Spaeth, and Lakhani (2003; tinyurl.com/atwpr4k) in the case of open source software community inflow. A warning sign for the sustainability of open communities is the apparent increased difficulty of recruiting and retaining new members, which has been observed for the Wikipedia project in particular (Ortega, 2009; tinyurl.com/ahhvu55). However, the diversity of projects makes it difficult to assess whether these concerns are justified and how broadly they apply. To have better evidence of the situation, we need to understand better how production is organized in such projects. However, comparing open source projects is complicated because they use different techniques (e.g., languages), different technical systems to support cooperation (e.g., version control systems), different management structures, and so on. In this article, we focus on the methodological aspect of the measurement of the efficiency to propose and validate a methodology before applying it to complex-to-compare projects. We therefore sought a setting that would provide a greater degree of comparability between projects.

Wikipedia (wikipedia.org) is an interesting setting for our research, for several reasons. First and foremost, Wikipedia is a large and successful online community, comparable in many ways to open source software projects. However, as noted above, it is also a project that has experienced an apparent slow-down in the recruitment of new editors, raising the question of sustainability. Second, the structure of Wikipedia lends itself to a comparison of efficiency. Wikipedia maintains a separate version of the encyclopedia in different languages. Each version has an independent collection of articles maintained by its own community of editors (though nothing other than language fluency prevents an editor from contributing to more than one Wikipedia language). Importantly, these communities of editors have reached different levels of maturity. Some communities are quite mature, whereas others are still getting started and yet others fall somewhere in between. However, they all share the same tool for collaborative editing (MediaWiki; mediawiki.org) and the same basic rules to guide this cooperative editing: the “five pillars” of Wikipedia (tinyurl.com/bhs3m). As well, if we measure the global structure of these communities as a network in which the articles are the nodes and the hyperlinks to other articles connect these nodes, it seems to be approximately similar, at least for the largest Wikipedias (Zlatic and Stefancic, 2011; tinyurl.com/andcqo7). In contrast to studies on open source software (e.g., Crowston et al., 2006: tinyurl.com/a3wec75; Koch, 2009: tinyurl.com/a74aqj7) that compare projects that use various technologies, programming languages or collaborative tools, this uniformity may help us to better understand which differences can be correlated with process evolution.

The article is organized as follows: first, we define the inputs and the outputs to be evaluated in our analysis of efficiency and our analysis approach, multiple-input multiple-output efficiency techniques (specifically data envelopment analysis). Then, we present the data and our current results. We discuss these results in the last section and present some conclusions that can be useful for practitioners, managers, and facilitators in these kind of open communities to assess their current evolution and prevent negative factors that could influence proper development of these communities in due course.

Theory Development

This analysis focuses on a comparison of the 39 largest Wikipedias (according to the official article count provided by Wikimedia Foundation, which is displayed on the home page of each version). The unit of analysis for our study is a Wikipedia community writing in a specific language (e.g., French, German, Japanese). However, we decided not to include the English Wikipedia in this analysis, because it is a prominent outlier regarding many aspects. It is by far the largest Wikipedia by number of articles, with four times more entries than the next language (the German Wikipedia), so we were concerned that it would have too great an influence on our results. It also exhibits a much broader community, attracting editors from the five continents, as it has become the default language version for many contributors and readers. As such, it is difficult to define the population from which English-language editors are drawn, which is a necessary step in our analysis.

Given this sample of projects, we assess the efficiency of the different Wikipedia communities in each language to turn their readers (inputs) into contributors (outputs). Research has shown that a mix of experienced editors and fresh newcomers increases the likelihood for an article to reach the top quality, or “Feature Article”, level in Wikipedia (Ransbotham and Kane; 2011: tinyurl.com/azbxulp; Bryant et al., 2005: tinyurl.com/a3h3d6x; Arazy et al., 2011: tinyurl.com/allx4j8). Thus, the output of the recruitment process is the number of editors (of different types, described below) contributing to the project. We take as input the number of potential contributors, also described below.

Economists formalize the link between inputs and outputs as a production function, literally a mathematical function giving the amount of outputs of a process for a given amount of inputs. Efficiency is outputs divided by inputs; to optimize efficiency means to obtain the maximum possible outputs for a given amount of inputs. In our case, the form of this production function is unknown, as are the coefficients relating its components. However, we are not trying to propose a characterization of the Wikipedia production function, but rather to evaluate if communities in different languages are more (or less) efficient than others. Following Farell (1957; tinyurl.com/b7apobr), the relative efficiency of different producers can be compared by examining the "frontier production function". This function describes, for various combinations of inputs and outputs, which producers are efficient. In other words, efficiency refers to the members of a sample of producers who have the highest outputs for a particular mix of inputs. Note that this definition of efficiency is relative rather than absolute; there is not some theoretical sense behind the term "efficiency". An additional consideration in analyzing the efficiency of production is the question of "return to scale", that is, whether a big project may be more efficient because of its size (e.g., in a larger and better known project, it is easier to attract new producers) or perhaps less efficient because of the overhead of coordinating more participants.

There are several techniques for estimating the frontier production function. A detailed comparison is out of the scope of this paper, but interested readers are referred to (Kitchenham, 2002; tinyurl.com/bgd2z4j) for a more complete discussion of these techniques regarding software production. We used the data envelopment analysis (DEA) models originally proposed by Charnes, Cooper, and Rhodes (1978; tinyurl.com/b2tuxpz), following Koch's (2009; tinyurl.com/a74aqj7) use in the case of open source software. Koch noted that "these models were developed to measure the efficiency of non-profit units, in which neither clear market prices for their inputs and outputs exist, nor a clear evaluation for their relations" (p. 403). In addition, "DEA can account for economies or diseconomies of scale, and is able to deal with multi-input, multi-output systems in which factors have different scales" (p. 398). These characteristics made DEA an appropriate technique for our comparison of different Wikipedia projects.

Data

External data (inputs)

To estimate the input to the recruitment process, we need data on the number of potential editors for each Wikipedia in a different language. We consider this group as the number of people with a tertiary education, who speak that language and have access to the Internet. The rationale for this choice can be found in Glott, Schmidt, and Ghosh (2010; tinyurl.com/66zazh5), as well as in a survey on the French Wikipedia (Dejean and Jullien, 2012; tinyurl.com/bz6x7zn), showing that Wikipedia contributors are significantly more educated than readers. To estimate the Internet population, we retrieved data from Internet World Stats (internetworldstats.com). This site aggregates Internet usage data from several sources, including "data published by Nielsen Online, by the International Telecommunications Union, by GfK, local Regulators and other reliable sources". Data are available at the language level for Chinese, Spanish, Japanese, Portuguese, German, Arabic, French, Russia, and Korean. For other cases (Dutch, Hungarian, Persian, Romanian, Bulgarian, Croatian, and Greek), we calculated the total number of users by multiplying the Internet rate in the main countries speaking the language by the population of these countries plus the population of minorities speaking the language by the Internet rate in the other countries where the language is spoken. A similar procedure has been conducted for the number of people with a tertiary-level education by language. The primary data for this measure comes from UNESCO (tinyurl.com/blx7n6f) for most of the countries in the study and the OECD for Russia (tinyurl.com/ahogygv) and China (tinyurl.com/b3ubalt). Of course, these sources provide only an approximation of desired input variables, but they are our best estimates. However, drastic inaccuracy in these estimates would in turn affect our productivity estimation.

Wikipedia data collection (outputs)

As in prior studies of Wikipedia (e.g., Wilkinson and Huberman, 2007: tinyurl.com/bjgge7x; Ortega et al., 2007: tinyurl.com/auwcneq; Ortega et al., 2009: tinyurl.com/bfb3spm), we relied on the database dumps published by the Wikimedia foundation. These databases contain complete records (date and time, author, etc.) of every single contribution that comes in the form of a "revision" to any page in any of the 39 Wikipedias under study. Thus, it is possible to count the number of active editors per month and break them down in three groups, following the definitions offered by Wikimedia Foundation (stats.wikimedia.org/EN/): very active Wikipedians (those with 100 or more revisions in a given month); active Wikipedians (between 5 and 100 revisions in a given month), and other contributors (those with fewer numbers of edits in a certain month). For this step, data extraction has been implemented as a software program that is part of WikiDAT (Wikipedia Data Analysis Toolkit; tinyurl.com/aykvdbt).

Analysis approach: DEA modelling
We must contemplate two main criteria regarding the choice of a DEA model: its orientation (input-oriented or output-oriented) and the return to scale in the production process. Regarding the first criteria, as in (Koch, 2009; tinyurl.com/a74aqj7), an output-orientation seems to be more appropriate given that, for a certain period of time, the inputs (the population of volunteers potentially joining a Wikipedia in a certain language) are more or less fixed and the goal is to maximize the output. As for the second criteria, considering the study on collective action (Marwell and Oliver, 1993; tinyurl.com/bapafxc), the analysis of software projects, and our previous discussion, it seems rather difficult to assume a constant return to scale. Instead, these projects seem to have an increasing return to scale in a first phase, and then a decreasing one. Hence, we use the BCC-O (output-oriented) model (Banker, Charnes, and Cooper, 1984; tinyurl.com/bxv62wy) that lets us assess the return to scale. For the data analysis, we adapted Sadiq's (2011; tinyurl.com/bxadd9r) macro under SAS.

Findings

An exploratory plot of our datasets shows a strong (but not perfect) correlation between the total number of Wikipedia contributors, the Internet population (Figure 1), and total tertiary-educated members of the population (Figure 2). Using the DEA model, we can identify different levels of efficiency in the conversion of these inputs to the Wikipedia community of contributors of different kinds. We first apply a constant return to scale model, then we introduce the possibility of a variation in return to scale. The results for this analysis are shown in Figure 3. The projects are listed in decreasing order of size. The bars indicate the relative efficiency. The longest bars, representing 100% efficiency, correspond to projects on the efficient frontier, that is, those that create the most outputs from their particular combination of inputs. Shorter bars represent projects that use a similar mix of inputs but produce comparatively fewer outputs than other projects. Specifically, certain Wikipedias, such as Malaysian (ms), Arabic (ar), and Chinese (zh), have many fewer editors than would be suggested by the population of Internet users who could become editors, whereas Estonian (et), Hungarian (hu), Norsk (no), and Finnish (fi) show high efficiency in recruiting editors. As far as the return to scale is concerned, Table 1 presents the sign of the return to scale variable. It seems that the largest and most efficient projects exhibit decreasing return to scale, suggesting increased difficulty in recruiting new Wikipedians. On the other hand, when they are efficient, the smaller Wikipedias seem to be still in an increasing return to scale phase.

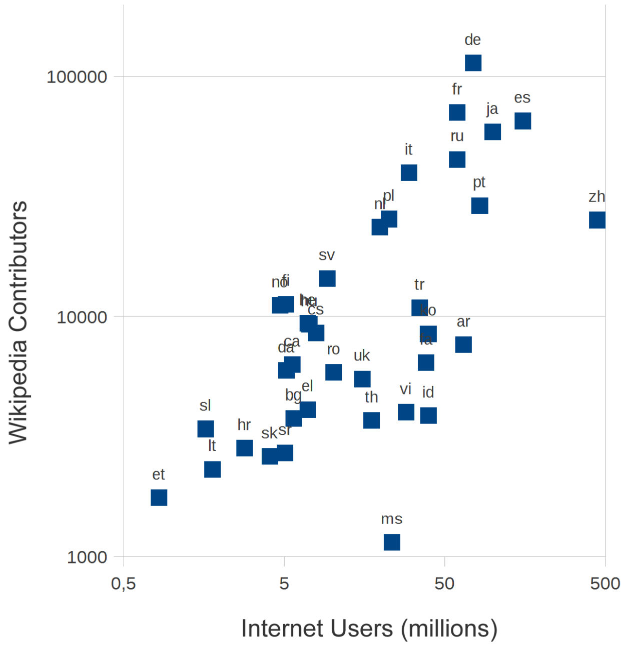


Figure 1. Number of contributors versus Internet population

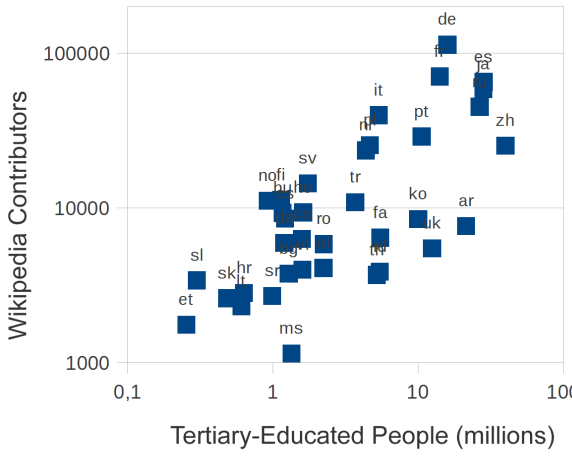


Figure 2. Number of contributors versus population with a tertiary education

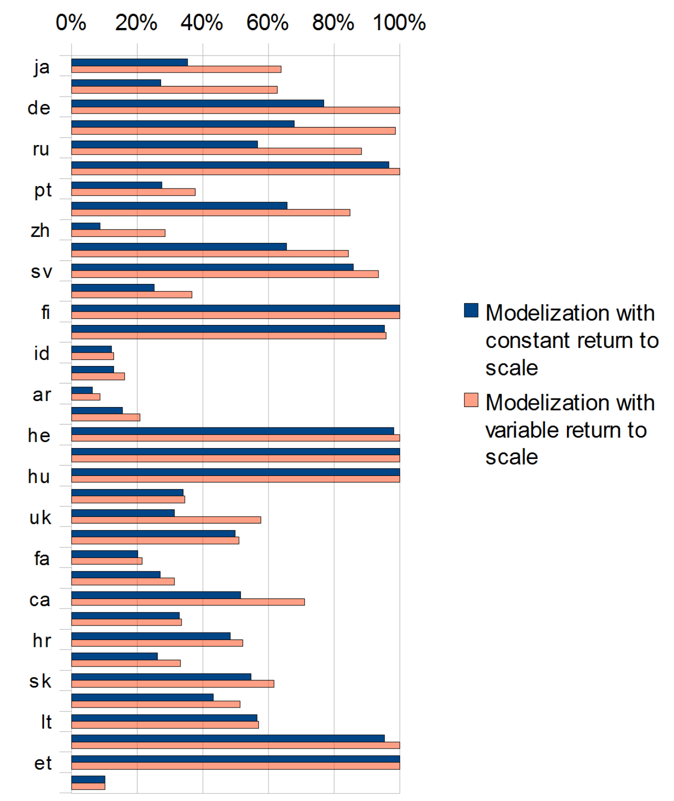


Figure 3. Efficiency in recruitment of contributors. (Projects are listed in decreasing order of size.)

Table 1. Return to scale for the recruitment of contributors. Efficient projects are highlighted in bold-italics and red font.

Project		
Japanese	ja	-1.57
Spanish	es	-1.60
German	de	***-0.04***
French	fr	-0.11
Russian	ru	-0.12
Italian	it	***-0.12***
Portuguese	pt	-0.17
Polish	pl	-0.14
Chinese	zh	-0.15
Dutch	nl	-0.10
Swedish	sv	-0.45
Turkish	tr	-0.29
Finnish	fi	***-0.03***
Czech	cs	-2.19
Indonesian	id	-0.65
Thai	th	-0.38
Arabic	ar	-0.73
Korean	ko	-0.09
Hebrew	he	***-0.08***
Norwegian	no	***0.02***
Hungarian	hu	***-0.14***
Vietnamese	vi	-0.36
Ukrainian	uk	-0.64
Danish	da	-1.31
Farsi	fa	-0.62
Romanian	ro	-0.15
Catalan	ca	-0.78
Bulgarian	bg	-0.49
Croatian	hr	0.34
Greek	el	-0.75
Slovak	sk	0.42
Serbian	sr	-0.16
Lithuanian	lt	0.13
Slovenian	sl	***0.15***
Estonian	et	***-0.19***
Malaysian	ms	-0.56

Conclusion

The work presented here provides an initial step to identifying differences in the work practices of the various Wikipedia projects, shedding light on the sustainability of such collective intelligence projects, and it proposes a way to extend the work initiated by Stvilia, Al-Faraj and Yi (2009; tinyurl.com/bdlounp), Hara, Shachaf, and Hew (2010; tinyurl.com/aajhf93), and Callahan and Herring (2011; tinyurl.com/b3pjrff). Our analysis indicates that the size and maturity level of the project matters, because the largest Wikipedias are assessed by this model as being inefficient (that is, recruiting proportionally fewer new editors for a given mix of potential participants than other projects with a comparable mix). If we add a factor to control for return to scale, the largest projects increase their performance, but display a negative return to scale. In other words, the larger projects are demonstrably in a phase where they are less able to recruit new members. Furthermore, the analysis reveals striking differences in efficiency among the smaller projects, which presumably are otherwise at similar states of development.

The results of our analysis are suggestive, but clearly represent just a first step. While we have shown differences in efficiency, we do not yet fully understand why these differences arise. The next step of the research will be to find better explanations for these differences. There can be many possible explanations for difficulties in recruitment, but research literature on open source software projects (Koch, 2008; tinyurl.com/b6hcrll) and on collective action more generally (Marwell and Oliver, 1993; tinyurl.com/bapafxc) suggests that such a slow-down may simply happen as a result of the project entering a mature phase in which fewer additions, and thus fewer contributors, are required. Nevertheless, a more troubling possibility is that the evolution of the projects has led to the development of working patterns that make contributing to these projects more difficult. This scenario could make participants' work less rewarding (Ransbotham and Kane; 2011; tinyurl.com/azbxulp), raising invisible barriers to contributions from outsiders and new members (to take Ostrom's perspective) and thus threatening the long-term sustainability of the project. Distinguishing these possibilities for the larger projects is important to understanding their prospects.

However, explaining the differences among the smaller projects requires more nuanced explanation. While the current data do not provide an answer, we hypothesize two possible explanations. First, many of the less efficient projects have a lower level of tertiary-educated people compared to the efficient group. This difference could be a key to explaining the low efficiency of recruitment. A second speculation regards the effects of control of information: many of the low-efficiency projects are tied to countries where the Internet and the production of information is more closely controlled by the authorities than in the efficient group. It may be that freedom of expression is pre-requisite for efficient recruitment of editors. Zhang and Zhu's (2011; tinyurl.com/afqraut) recent study on the Chinese Wikipedia gives arguments for this hypothesis.

Better understanding these differences should provide insight for the long-term sustainability of both Wikipedia as well as other open knowledge-creation projects. In particular, the first hypothesis suggests that these projects are dependent on the investments made in education by the countries in which the projects are situated. Given the importance of the tertiary education variable, universities seem to be appropriate places to promote Wikipedia, which is in line with the Foundation's strategy regarding Wikipedia Education Program (tinyurl.com/9cqrh3r).

Another topic for future research is to address the limitations in the current study. A main limitation is that the validity of our analysis is dependent on the quality of the data used. In particular, the external data used for the inputs to the recruitment process are only best estimates. Systematic errors in these data would affect our measure of the relative efficiency of recruitment for the affected languages. On the other hand, while we are quite confident in the data extracted from the Wikipedia dumps, a limitation of the work presented here is that we evaluated the projects only for a single month, August 2011. Having only one month of data could lead to misinterpretations, especially taking into account that August is a vacation month in some countries. We are working on extending the analysis to twelve months and doing a mean estimation of the efficiency of the various projects.

Future research might also examine the transferability of the proposed methodology to open source software. In characterizing the open source production function, characteristics from the software engineering perspective such as the time to close bugs, the number of issue reports submitted, or activity in the mailing lists, may be of equal importance to the total number of contributors for evaluating the sustainability of a project. It is also important to consider the age of the project, as reflected in the return to scale effect. A complication we noted in the introduction to the article is that the diversity of open source projects makes it hard to compare them. One possible approach would be to compare different sub-projects within a larger project, which might control for variability in tools and processes.

Sustainability of Open Collaborative Communities: Analyzing Recruitment Efficiency

Kevin Crowston, Nicolas Jullien, and Felipe Ortega

About the Authors

Kevin Crowston is a Distinguished Professor of Information Science at the Syracuse University School of Information Studies (aka the iSchool). He is currently on a temporary rotation as a Program Director for the Human-Centered Computing Program at the US National Science Foundation in the Information and Intelligent Systems Division of the Computer and Information Science and Engineering Directorate. His research examines new ways of organizing made possible by the extensive use of information technology.

Nicolas Jullien is an Associate Professor at the LUSSI Department of Telecom Bretagne (Brest, France). His research interests are on the organization and the attractiveness of open, online communities (Linux, Wikipedia). Most of his papers are available at: tinyurl.com/asfqzsm

Felipe Ortega is a Researcher in the Department of Statistics and Operations Research at University Rey Juan Carlos in Madrid, Spain. He is also a part-time Associate Professor at University Alfonso X El Sabio, teaching courses in the Information and Communication Technologies Department. His research is focused on open online communities, with emphasis on data retrieval, replicability, and data analysis.

Citation: Crowston, K., N. Jullien, and F. Ortega 2013. Sustainability of Open Collaborative Communities: Analyzing Recruitment Efficiency. *Technology Innovation Management Review.* January 2013: 20-26.

Going Open: Does it Mean Giving Away Control?

Nadia Noori and Michael Weiss

Many people think that open source projects are sort of chaotic and anarchistic. They think that developers randomly throw code at the code base and see what sticks.

Mitchell Baker
Chairman, Mozilla Foundation

Open source software has evolved from being an effort driven by a collective of volunteers to become an integral part of commercial software. Constant demands for new features besides maintaining product quality made companies seek open source as an answer for these demands. These growing demands brought with them control of quality, architecture, contribution management, and community management.

This article explores the governance strategies adopted by open source software projects to manage the quality of complements (such as plug-ins that extend a platform's functionality) developed by community members outside the core team. The outcomes of the research contribute to our understanding of the strategies followed by different open source platform owners (the open source project initiators) to manage external innovation in the case of platform extensions in two areas: i) governance models and ii) regulatory tools.

Introduction

Conventional research on open innovation and collaboration examines how organizations open up the innovation process and how they control the collaboration with others. The research also identifies the need for governance models and architecture of participation in collectives that embrace open innovation practices to help maintain momentum and ensure continuity. Existing research on systemic innovation and platform ecosystems helps us understand the structure of platform-complement product systems (Boudreau and Hagiu, 2009: tinyurl.com/a3xc7xj; Baldwin and Woodard, 2009: tinyurl.com/aceg9ac; Dahlander and Gann, 2010: tinyurl.com/chacrs9). Studies examine the types of relationships and information transactions among the members of a platform ecosystem, the importance of platform owners as regulators of the ecosystem, and regulatory instruments available to them.

Iyer (2006; tinyurl.com/csmescv) describes how software companies are operating in a small world of interconnected networks and how innovation is increasingly taking place in such networks. The question is no longer whether or not to open the innovation process and collaborate, but how to best leverage a network of external parties (Tuomi, 2002: tinyurl.com/crbmhrv; Moore, 2006: tinyurl.com/5rtbj6u; Pisano and Verganti, 2008: tinyurl.com/67bcd3b; Vujovic and Ulhoi, 2008: tinyurl.com/b6l3jov).

In most open source projects, there is a focal organization that acts as that platform owner (or keystone in a project ecosystem) that provides the platform and facilitates contributions by other members in the community (Iansiti and Levien, 2004: tinyurl.com/7t4xgvn; Noori and Weiss, 2009; tinyurl.com/ae2n8su). A platform can be a product, service, or technology that provides a foundation for other parties to develop complementary products. The platform can be owned by a single player, as in the case of Apple's control over the iPod and iPhone application ecosystems, or it can be developed and influenced by a group of players, as in the case of the Eclipse software development platform (des Rivieres and Weigand, 2004: tinyurl.com/arwl3j3; Shuen, 2008: tinyurl.com/anvzq6y; Hagiu and Yoffie, 2009: tinyurl.com/bu7zn3n).

The literature on how platform owners manage complementary markets is focused on complements that build on the platform but not on complements that integrate with the platform, such as extensions. Platform extensions extend the functionality of a platform beyond its core capabilities and can integrate at different levels with the platform and other existing extensions, and this integration reflects upon the integrity of the platform. So, the quality of platform extensions contributed by community members has become a concern for platform owners because it can comprise the overall quality of the platform and other complements as well (Messerschmitt and Szyperski, 2003: tinyurl.com/d2ulug3; Gawer and Cusumano, 2008: tinyurl.com/bjkhq3j; Bosch, 2009: tinyurl.com/arrlwcz).

We studied six open source projects (Eclipse, Firefox, Apache HTTP, Spring, OpenOffice, and MySQL) to examine the strategies followed by the platform owners to manage contribution by external parties and to manage the quality of the complements developed by those parties. We limited the research to one type of complement, platform extensions, because of the direct effect of the quality of extensions on platform integrity as well as to keep the study to a manageable size.

Research Method

Research on regulating the platform extension development process is still in its early stages, therefore there is little known about strategies adopted by platform owners to control the quality of extensions developed by external parties (Hagiu, 2009; tinyurl.com/adn8gb2). We used the case study research approach due to the novelty of the research area (Eisenhardt, 1989; tinyurl.com/7dfuc3z).

In our research, we examined 12 open source platforms and related commercialized software platforms (if any) from the time period between 2000 and 2009. The unit of analysis was a software platform that provides the ability for external parties to extend its functionalities. An example of an extension we examined is the tabbed browser extension for Firefox, which affects the core functionality of the platform.

Eisenhardt (1989; tinyurl.com/7dfuc3z) recommends theoretical sampling when selecting cases for case study research, which implies that cases may be chosen to replicate previous cases or extend emergent theory. Unfortunately, previous research did not provide a reference for selecting cases using theoretical sampling; therefore, data was collected in two initial waves of four cases each and, once preliminary results emerged, a third wave of four more cases was chosen using theoretical sampling.

Governance Models

Within the sample cases, we found three types of governance models: tight-control, loose-control, and hybrid-control. Each governance model consists of the following attributes: community structure, extension types, and governance structure and network openness. Each model was associated with non-trivial trade-offs in terms of governance, openness, quality, and flow of ideas.

Furthermore, different levels of governance and openness may be applied to different types of extensions. Internal extensions, which are more widely used and usually deployed together with the platform core, are often more tightly controlled than external extensions, which are developed to meet more specialized needs. In the case of internal extensions, there is a significant impact of low quality of those extensions on the platform and on each other; a reduced flow of new ideas is traded off against higher quality. As for the external extensions, it is more important to allow new ideas to develop than to monitor their quality. Yet, the distinction between internal and external extensions is not fixed; over its life, an extension may change its type.

The governance structure of a network can be either hierarchical, flat, or a hybrid between these extremes. In a hierarchical governance structure, the platform owner both defines problems and selects which solutions are adopted, whereas in a flat structure a community decides on both problems and solutions. The case between those two extremes is a hybrid of hierarchical and flat structures: although the community decides on problems, the platform owner selects solutions.

Openness of a platform network refers to the degree to which participation in the network is open. In an open network, any party (partners, customers, or even competitors) can contribute to the platform. Open source projects are examples of this type of network. In a closed network, the platform owner selects who can participate based on the capabilities and resources required for the innovation (Pisano and Verganti, 2008; tinyurl.com/67bcd3b).

Table 1 summarizes the governance models, their attributes, and their associated effects on the quality of extensions and flow of ideas. Table 2 summarizes the advantages and disadvantages of each of the three governance models with an emphasis on the quality of extensions and flow of ideas.

Going Open: Does it Mean Giving Away Control?

Nadia Noori and Michael Weiss

Table 1. Summary of governance models and associated extensions types, governance structure, and network openness

Governance Model Type	Community Structure	Extension Types	Governance Structure	Network Openness	Extension Quality	Flow of Ideas
Tight-control model	Core and internal	Internal only	Hierarchical	Closed/ open	Medium to high	Low to medium
Loose-control model	Core and external	Internal	Hierarchical	Closed	Medium to high	Medium
		External	Flat	Open		
Hybrid-control model	Core, internal, and external	Internal	Hierarchical / flat	Open	High	Medium to high
		External	Flat	Open		
		Commercial (internal or external)	Hierarchical	Close		

Table 2. Advantages and disadvantages of the three types of governance models

Governance Model Type	Advantages	Disadvantages
Tight-control model	• Easy to control the development process • Easy to control the quality of the product • Direct efforts towards achieving the platform owner's desire to meet certain requirements	• Limited number of ideas
Loose-control model	• Unlimited number of ideas • Unpredictability could lead to unusual use of the platform functionalities that can trigger innovation in the community	• Unpredictable community of developers • Difficult to control the community of developers • There is no way to guarantee the quality of the product
Hybrid-control model	• Platform owners have control over who develops what in the community • Platform owners can control the quality of developed product on different levels • Platform owners are the orchestrators for the flow of ideas in the community	• Requires effort from the organization to oversee the different parties involved

Regulatory Instruments

To support the governance models and enforce control over the development process of the platform extensions, the platform owners needed to use a number of regulatory instruments in conjunction with the governance model. In our reseach, we found that regulatory instruments enabled platform owners to manage information transactions in the platform network, support the process of developing extensions, and control the quality of the product by providing the community with tools to develop, test, and integrate extensions, and to share the experience among the platform members.

The regulatory instruments help platform owners to establish barriers of entry to the developer community such as pricing or the development process. Instruments such as platform architecture and toolkits help to create technical boundaries between the platform and the developer community.

Regulatory tools such as pricing and membership refer to a pricing schema set by the platform owner to charge third-party developers for gaining access rights to the platform, information, and service. Pricing and membership programs enable platform owners to filter the inflow of ideas as well as the quality in the network. Although using the pricing instrument enables the platform owner to control the flow of ideas, there is no guarantee of eliminating poor-quality contributions because it depends on the pricing structure and the platform owner's need to access external resource (Hagiu, 2008; tinyurl.com/bymusad).

Development toolkits are another form of regulatory tool. Toolkits are a combination of software infrastructure and development frameworks that reduce the time and effort required to develop, provision, and operate extensions; they also contribute to quality. For example, to ensure the quality of extensions, the Mozilla project offers a toolkit consisting of several tools that include a testing framework to test the performance and quality of Firefox extensions. Also, the Eclipse platform provides a plug-in development environment, a comprehensive series of build and user-interface creation tools, and tools for API maintenance.

Sandboxes are another type of regulatory tool in which extension developers are allowed to test their extensions in the actual deployment environment. For example, Firefox provides a sandbox review process on its Firefox add-on site, where extensions are available for trial and testing by the community. The sandbox review process enables the developers to test their extension before moving it to the general-availability phase.

Introducing a development process is also another regulatory tool used by platform owners to control the quality of developed extensions and filter the inflow of ideas into the platform ecosystem. An example of the development process used as a regulatory instrument is the incubation process, which is another method used by platform owners to control the quality of extensions developed by external parties. The incubation process enables the platform owners to filter the flow of ideas in the internal-extensions community of contributors (Duenas et al., 2007; tinyurl.com/aytv5x7). For some platforms, such as Mozilla, the incubator is a working directory that is considered a testing ground for experimenting with new ideas and it is a workspace where lead developers or module owners work with inexperienced developers.

Practical Implications

The results of our research are relevant to managers of both open and closed source platforms, third-party developers creating platform extensions, and researchers in innovation management. The research provides collaboration models that help platform owners understand the strategies adopted by other platform owners to manage the quality of platform extensions.

The models are a combination of collaboration governance structures and regulatory tools that helped platform owners to leverage the innovation process in their ecosystems and provide guidelines for developing third-party complements. The research also opens opportunities for future research on creating models for how platform owners can maximize user innovation in platforms, and how they can manage the platform-extension development process.

Conclusion

Open source is a living example of the viability and sustainability of the open-innovation model. The process of going open and maintaining growth and success of the open source platform is not chaotic or a set of random actions. Throughout the years, the open source community has learned how to organize itself and provide collaboration models and tools that fit within the free/libre open source software context. These communities needed such control mechanisms in place to ensure quality and maintain growth. Open source platforms had evolved from voluntarily initiatives to sustainable entities alongside commercial equivalents inside the software industry.

Going Open: Does it Mean Giving Away Control?

Nadia Noori and Michael Weiss

About the Authors

Nadia Noori holds an MASc degree in Technology Innovation Management from Carleton University in Ottawa, Canada. Her research interests includes open source platforms, governance models and collaboration frameworks, and product architecture and design. She works currently as a Production and Platforms Manager at Coral CEA, a not-for-profit organization based in Ottawa.

Michael Weiss holds a faculty appointment in the Department of Systems and Computer Engineering at Carleton University, and is a member of the Technology Innovation Management program. His research interests include open source ecosystems, mashups/Web 2.0, business process modeling, social network analysis, and product architecture and design. Michael has published on the evolution of open source communities, licensing of open services, and the innovation in the mashup ecosystem.

Citation: Noori, N. and M. Weiss. 2013. Going Open: Does it Mean Giving Away Control? *Technology Innovation Management Review*. January 2013: 27-31.

The Evolving Role of Open Source Software in Medicine and Health Services

David Ingram and Sevket Seref Arikan

“*In attempting to arrive at the truth, I have applied everywhere for information but in scarcely an instance have I been able to obtain hospital records fit for any purpose of comparison. If they could be obtained, they would enable us to decide many other questions besides the one alluded to. They would show subscribers how their money was being spent, what amount of good was really being done with it or whether the money was not doing mischief rather than good.*”

Florence Nightingale (1820–1910)
Founder of modern nursing

The past five decades have witnessed immense coevolution of methods and tools of information technology, and their practical and experimental application within the medical and healthcare domain. Healthcare itself continues to evolve in response to change in healthcare needs, progress in the scientific foundations of treatments, and in professional and managerial organization of affordable and effective services, in which patients and their families and carers increasingly participate.

Taken together, these trends impose highly complex underlying challenges for the design, development, and sustainability of the quality of supporting information services and software infrastructure that are needed. The challenges are multidisciplinary and multiprofessional in scope, and they require deeper study and learning to inform policy and promote public awareness of the problems health services have faced in this area for many years. The repeating pattern of failure to live up to expectations of policy-driven national health IT initiatives has proved very costly and remains frustrating and unproductive for all involved.

In this article, we highlight the barriers to progress and discuss the dangers of pursuing a standardization framework devoid of empirical testing and iterative development. We give the example of the openEHR Foundation, which was established at University College London (UCL) in London, England, with members in 80 countries. The Foundation is a not-for-profit company providing open specifications and working for generic standards for electronic records, informed directly by a wide range of implementation experience. We also introduce the Opereffa open source framework, which was developed at UCL based on these specifications and which has been downloaded in some 70 countries. We argue that such an approach is now essential to support good discipline, innovation, and governance at the heart of medicine and health services, in line with the new mandate for health commissioning in the United Kingdom's National Health Service (NHS), which emphasizes patient participation, innovation, transparency, and accountability.

The Evolving Role of Open Source Software in Medicine and Health Services

David Ingram and Sevket Seref Arikan

Introduction

The quotation from Florence Nightingale, a pioneering advocate for a statistical approach to the study of healthcare services, serves two purposes in focusing this article. First, although from a time long past, it resonates tellingly with contemporary critical concerns about the efficiency and effectiveness of health services. Second, it makes a connection between health records that exist to document and make accountable the care given by a professional team to a patient, and the external overview and governance of those services for populations of patients. There is an unavoidable and continuous tension between the focus on the care of a particular patient, in a particular context over time, and a smoothed out picture of the care of a population of patients in a diversity of contexts. A burden of data capture is always imposed, which too often adds cost and pressure to professional practice but fails to deliver value in terms of efficiency of workload and improvement in outcomes achieved for patients. Data are extracted from medical records or collected again, mapped and codified, analysed, and represented in different contexts. At each stage, there is potential for clinical requirement to become blurred and provenance of data to become obscured. This may then reflect in poor design and implementation of software, and in the integrity of systems, creating potential for undue confusion, cost, and error.

In the 60-plus years of the National Health Service (NHS) in the United Kindom, medicine has moved from a small-volume, low-profile, trusted, and generally accepted service to a high-volume, high-profile, critically scrutinized service. The breadth of services now provided within the healthcare system is very considerable, and their focus ever changing. From an information viewpoint, taking a snapshot as things are now, one might observe that:

1. Biomedical science is being transformed.

2. Healthcare and research are increasingly technology and information intensive.

3. Multiple legacy information systems are in use to support and link healthcare, research, and industry.

4. Governments now want pervasive and standardized ICT infrastructure for healthcare.

5. Many other initiatives in the commercial, public, and social enterprise and voluntary sector domains are creating relevant infrastructures and de facto standards.

Taking another snapshot of the experiences of patients in their encounters with advanced healthcare systems, Blendon and colleagues (2003; tinyurl.com/bbm8z76) conducted a large, multinational survey of patients with chronic conditions. In the United Kingdom, they found that:

1. Two-thirds of patients surveyed were not engaged in discussion about their own treatment and care, 40% did not have goals of treatment made clear, and 20% received conflicting information from different professionals.

2. Twenty percent of patients were victims of medical error in the past two years (9% with serious consequences).

3. Thirteen percent of patients were sent for duplicate tests and half had to repeat health history for different professionals due to medical records not reaching consultations on time.

The challenge of achieving an electronic health care record capable of representing and sharing the content and meaning of the treatment and care of a patient, or groups of patients, within their family and social contexts, has been tackled by many luminary clinical champions and pioneers, starting with Octo Barnett and Howard Bleich in Boston (Massachusetts General Hospital and Beth Israel Hospital) and John Anderson at King's College Hospital, London, in the 1960s.

The King's College Hospital project was the first attempt in the United Kingdom to computerize patient records in the NHS. The project, under the auspices of a brave and innovative clinician, then the Professor of Medicine at King's, was funded as an experiment in meeting what were likely, at the outset, to have been considered well-understood clinical requirements, using well-established computer technology. It was commissioned by the then NHS Supplies Division. In that era, there was almost nothing by way of digital imaging, fast international computer networks, nor even standard database methods – a 5 megabyte disk cartridge was a bulky item. We have learned a lot since then about the problems of clinical data management, and there have been huge technological advances, rendering much of everyday practice today highly dependent on information technology, with considerable convenience and benefit to all.

As the quotation from Florence Nightingale highlights, the quest to compile records that capture the essence of what medicine does, and how well, goes back much further. But, the capture and communication of clinical notes and histories is apparently still not yet computable, at scale, within the NHS, except within communities that share systems. And, of course, clinical and other requirements for managing such data have moved on a very long way.

We need to pause to reflect on the underlying reasons for the repeating failure and also on the implications of the upcoming changes towards what is being called personalized medicine, in which patients and their families will expect to be informed and to participate much more fully. Past failures must be owned at all levels of policy, profession, and practice. The underlying causes are uncomfortable because they highlight large and unrecognized gaps between the aspirations and hype that policy documents sign up to, and the knowledge, capacity, and new learning that are required to achieve useful ends. This is not just about practical expertise in design and implementation of computer systems. It is also about professional capacity to frame, deliver, and sustain realistic information systems and services that meet requirements for integrated and shared health and social care services, and that can and are made to work, for and by professionals, the patients themselves, and the industry.

The rapid pace of advance in the biosciences has required major research focus on new numerical methods, software tools, and data repositories, which has just about enabled that community to keep pace with the rapidly escalating demands in the context of services that explore and exploit genomics science. And, this is mainly in world-leading specialist centres, working and sharing approaches closely together, with their highly capable and expensive research and development teams.

The challenges implicit in achieving coherence of records within wider health services are also immense, but receive less academic and research profile than they merit because they have to involve heavy engagement with more typical everyday healthcare. In addition, when designing to meet data management requirements across diverse healthcare services, an industrial and scalable model for delivery of the needed IT services, employing suitably standardized and generic data architecture, is inescapable. To its credit, the most recent NHS National Programme for IT initiative (NPfIT; tinyurl.com/36sgmdp) took on that challenge, but for complex reasons, it substantially failed.

In the following sections, we introduce two key drivers for a more open approach to software within core systems supporting medicine and healthcare services, arising from the requirements for both technical and semantic interoperability. These requirements are the need for a practically informed approach to common data standards and the need for greater rigour and transparency in the way that systems are designed, tested, and held to account within wider clinical and information governance. We argue that these requirements can best be served within the context of open source communities of practice. We then introduce the openEHR Foundation and an open source implementation of its specifications, called Opereffa (OPEnEHR REFerence Framework and Application), as examples of initiatives pursuing these principles that have gained wide currency and adoption worldwide.

Data Standards

A fundamental challenge for healthcare IT is in marrying innovation led by scientific research and technological advance with innovation led by requirements for effective and efficient delivery of services that are valued by patients. The standardization of data and computer systems that work and can be sustained over time in these twin contexts is a considerable challenge. However, the challenge can and will, over time and with appropriate combinations of skills and resources, be overcome if it is properly framed. The quest for this standardization has been both albatross and Achilles' heel of NHS IT programmes for too long, the more so because, in some circles, a standard is seen as an enforced technical conformity of systems, as opposed to its primary purpose, which is to serve as a working language that enables and facilitates communication about meaning. This language will evolve over time, as clinical practice, science, and technology move forward, but therein lies the rub. For industry, control of such standards is an important insurance policy guaranteeing marketability of products. Standards are fought over and defended, becoming like tablets of stone, because software costs money and change in underlying standards can render systems rapidly unmarketable, unsustainable, and obsolete.

In recent decades, the difficulty of keeping pace with changing technology and infrastructure, both hardware and software, at scale, within large health systems, has proved unmanageable. So much so that the field has, in the main, been characterized by local successes (local, that is, to a particular clinical domain or institution delivering healthcare) and costly global failures (global in terms of sharing records across domains in which the data about a particular patient needs to be communicated and worked on, wherever they are cared for nationally or increasingly, internationally).

New approaches are needed to the challenge of defining practical and deliverable scope for standardization across health systems. These approaches need to focus on working more effectively and practically with professions, research, and industry, to learn, through practical implementation experience, about standards that work. The evidence of past endeavour is that the need and urgency of this goal is fully recognized, but the means of achieving it has, to date, been beyond health service, profession, and industry capabilities. As a result, our options when buying IT are too often locked down in inflexible designs, which cannot be changed because their technological underpinnings are already effectively obsolete, and where there has been too great a past investment committed in them to consider replacing them with further untried and untested approaches.

Towards Open Methods, Design, and Governance

In common with other developed health economies, the NHS is now embarking on the fourth major attempt to conquer this moving frontier. Other countries and large scale health providers have lost their billions, too. It seems a good time for the debate to focus differently, and we would suggest three general areas of concern.

1. **Discipline**: the computational science and professional skills needed for collecting, managing, and communicating high-quality clinical and biomedical data. This is a central but relatively underdeveloped discipline, as yet.

2. **Design for adaptability and change:** the principles underlying computer systems where data management can be locally customized and capable of evolving over time to meet new requirements and maintain lifetimes of analyzable clinical data, confidentially.

3. **Governance:** bringing a citizen focus to the overseeing of choices made about how personal health data is collected, shared and used in a manner that is more open and responsible about what we all, as citizens, may reasonably expect and over which we can, collectively, feel ownership and control.

These areas of concern relate directly to the rationale for requiring transparency of systems and for enhancing the prospects for progress in the domain through promotion of an open source approach to software, notably in key areas of shared information infrastructure where interoperability of systems is essential.

The Rationale for Open Source Implementation

This is not the place to analyze or present the breakdown that has occurred between clinical requirement, technological innovation, and institutional adoption and standardization of health records. Key barriers to progress have been and remain:

1. Lack of fit-for-purpose data standards

2. Failure to differentiate primary data requirements (for supporting the delivery of services at the coalface of care) from secondary data requirements (for supporting public health services, audit of health care delivery, and research)

3. Divergence of global and local requirements

4. Governance and confidentiality/privacy concerns

5. Sustainability over patient lifetimes

6. Multi-level, competing initiatives that lack a common strategy

7. Restrictive intellectual property protection; much that needs to be openly shared, debated, and learned from is hidden from view

The essential learning from past failure is threefold:

1. Overly centralized approaches are inevitably remote from the everyday realities of designing and delivering clinical services. These approaches rely too heavily on industry-derived solutions and have failed to conquer the problems of the domain.

2. Most, if not all, successful innovation in the field has rested on the shoulders of local clinical pioneers, who were ambitious, capable, and committed to the hard work of: i) learning by doing; ii) balancing the different drivers and constraints of medicine, technology, and organizational management; iii) plugging gaps in supporting infrastructure; and iv) keeping focused on clinical need.

3. Local communities of practice and industry working in partnership need shared methods and recognized discipline. These groups are charged with providing and sustaining information services, and they must be enabled and supported to learn and adapt.

In earliest times, many local health providers built local team capability in the domain. This was costly and placed a huge pressure on these teams because the requirements of the domain as a whole were only slowly being clarified as they did their work. It is in fact a classic "wicked problem", as characterized by Rittel and Webber (1984; tinyurl.com/a48syez) because:

1. There is no clear ownership of the problem, permission to experiment, or right to judge.

2. Working towards a solution clarifies and revises understanding of the problem.

3. The problem does not have right answers; it requires "good enough" approaches.

4. The problem is never completely solved.

5. Solutions require changes in behaviour of stakeholder groups.

6. The manner in which the problem is tackled is as important as how it is tackled.

Open source commends itself as an approach, not because in some magical way it makes it cost-free – that is an almost laughable simplification only meaningful to those who wish to exploit other peoples' efforts rather than join in with and learn from them, adding value back into the commons. Key features of an open source ecosystem that could contribute most towards progressive solution of the wicked problem are:

1. The promotion of effective and efficient innovation within software development communities. Clinical pioneers have traditionally had to build whole local infrastructures, and unnecessary efforts to "reinvent the wheel" are still common.

2. The pooling of development and maintenance costs for essential infrastructure

3. The enablement of the research interface. Discipline grows through sharing, testing, and reviewing methods.

4. Increased awareness and understanding of the inner workings of systems to improve procurement

5. The integration of systems and services and continued efforts to combat fragmentation

6. The enablement of more patient-focused services that can be managed and regulated more effectively and transparently. The goal is to create a more trusting environment of professional practice, patient participation, and public engagement.

But, there must be a business case for this approach. Support from government and industry, as well as health professionals, is needed for the transition to a viable open source framework and community.

openEHR

The challenge of the electronic healthcare record has been progressively understood and refined, in information terms, to embrace both the narrative and quantitative representations of knowledge useful to describe patients and their clinical and healthcare problems, the interventions through which they are cared for, and the outcomes achieved over time. To address this challenge, the openEHR Foundation (openEHR.org) was founded as a not-for-profit company. It builds upon earlier attempts to formalize information architecture for an electronic patient record infrastructure. For full details on the origins of openEHR, see: openehr.org/about/origins.html.

In simplest terms, openEHR provides several levels of design to help enable better more sustainable electronic records:

1. A domain reference model comprising a formally rigorous, openly change-managed set of building blocks that are needed to capture, organize and communicate the content of all health records, and the structures needed such that they can meet the wider ethico-legal requirements for aggregation and sharing of records.

2. An archetype model, open source software tooling, and a growing repository of published artefacts, supporting the ways in which these building blocks can be combined to build a clinical record. Each concept, such as a laboratory test result or an instruction to initiate a treatment or document its application, is

modelled by clinicians in ways that represent its generic sense and meaning (as an openEHR archetype – defined as a formal constraint on the reference model), for everyone working with it.

3. Customized local use of archetypes through templates (defined as a further constraint of the archetype), universal querying of the resultant record content, and a growing body of open source software "plumbing", to enable integration within the principal generic platforms on which software infrastructures are built.

The quest for a universal electronic health record has collapsed several times, about once every decade, in large part due to overly ambitious global and top-down programmes and initiatives, which lost touch with or ended up inhibiting or suppressing coal-face innovation by local champions. openEHR has sought to champion open specifications and an inclusive community of "doers" – people motivated principally to help necessary change happen. Individuals in this community are somewhat free-spirited, and thus sometimes a bit quarrelsome and disruptive, but none the worse for that, because the community has also sustained a largely united common vision and mission. It is now working internationally to evolve a business plan that can sustain its fundamental openness, empiricism, and inclusivity.

The architecture has been experimented with and adopted by individuals, small companies, universities, whole institutions (e.g., the Marand/Ocean openEHR-based paediatric centre patient record in Slovenia), and countries (e.g., Brazil). Spanish and Japanese language versions are available, and openEHR now has members in more than 80 countries, with progress underway towards regional chapters. It is gradually attracting the support of larger and more influential stakeholders. The Clinical Information Modelling Initiative (CIMI), which is centred in the USA and which dominates present markets for health IT, recently voted to base its work towards standardization of EHR content on the reference model, archetype formalism, and current shared CKM archetype repository of openEHR.

Opereffa

There are many gaps to be filled in making an open source framework for health information systems and services a reality. At the same time, as judged by the several hundreds of clinical database systems, collecting and analyzing clinical data for research in a typical teaching hospital trust, there is a huge unmet need for workbenches and tooling to help a new generation of innovators create the systems required for tomorrow.

The rationale for the Opereffa open source framework initiative (tinyurl.com/b3t38m7), currently being undertaken at UCL by its spinout company Charing Systems, is as follows:

1. Data mining tools and frameworks focus on system designs that are traditional now for information storage and retrieval, and that embody relational databases with normalized tables. Electronic health records are rarely implemented with such normalized designs, though the relational database is still the most widely used underlying technology.

2. These tools and frameworks are expensive, especially if they must be scaled.

3. Open source development frameworks help with the cost, but they are still "tools for the few". And, there is still very little open source tooling available to implementers to help them build standards-driven clinical information systems.

4. The Eclipse-based Opereffa framework is an open source test bed developed in an attempt to deliver a flexible information repository with rich methods of connectivity to other platforms.

Opereffa has been downloaded 1000 times in 70 countries and has featured in many local systems developments, such as a national surveillance system for tuberculosis in Cambodia.

Opereffa is working to:

- leverage high-performance, open source development frameworks

- explore requirements for simultaneous support of clinical service delivery and machine learning

- explore design approaches that can simultaneously improve information retrieval and machine intelligence performance

- develop richer, smarter methods of access to clinical data to better support machine intelligence, especially by delivering alternatives to SQL-based access to relational databases

- eliminate complicated bridges between clinical information systems and research data repositories

- explore new design and implementation patterns for the openEHR specifications that enable these goals

This all boils down to standards-driven persistence and data access: Opereffa and the openEHR Archetype Query Language (AQL). The importance of AQL lies in its capability to extend the usability of domain concepts used for modelling clinical data, into coherent querying of ensembles of clinical data. This approach goes beyond avoidance of impedance mismatch between object oriented and relational views of data; it converts the underlying domain models into a platform that supports all data access and processing requirements, with one coherent method.

The Opereffa architecture is illustrated in Figure 1. It uses:

1. Proven open source persistence stacks, aligned along the scale axis from PostgreSql to Hadoop

2. High-performance open source parallel processing frameworks for scaling upwards: Akka, Hadoop

3. Open source tooling to eliminate complicated technology and infrastructure management processes: the Eclipse framework

The goal is to bring all these technologies together within the strongly model-driven approach of openEHR, to achieve outcomes that are portable to other domains.

Conclusion

Building something that must, by its very nature, be a shared discipline, requires development of tried and tested methods. This cannot happen if those methods remain hidden away in systems, out of reach of critical appraisal and opportunities for shared learning. Openness and transparency about these methods is thus crucial. Intellectual property rights must be respected but there can be no discipline for others to build on if the methods underpinning the design and standardization of systems is not clearly expressed, validated, and made accessible and open to improvement in wider communities of adoption and practice.

Adoption of the openEHR specifications, and associated open source implementations such as Opereffa,

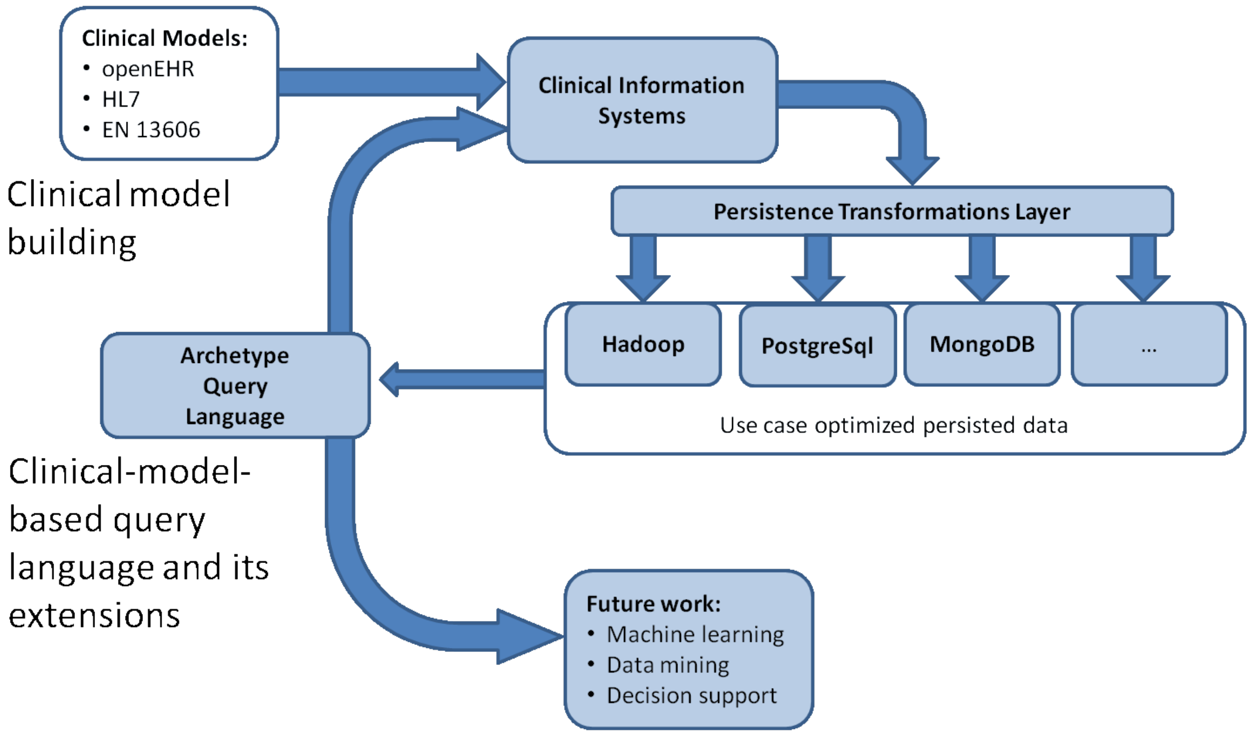


Figure 1. The Opereffa architecture

have succeeded as exemplars of good practice in a number of ways:

1. They are vital steps for model-driven life and clinical sciences data management:, i) analysis of domain data and processes, ii) building of computable information models, iii) implementation of model-driven functionality, and iv) avoidance of the tendency to "reinvent the wheel".

2. The archetype methodology (two-level modelling) of openEHR is reusable across a considerable variety of domains.

3. Tooling and computation is driven by one common infrastructure, which is, in turn, built on the reference model (RM).

4. The openEHR two-level modelling approach provides access to 20 years of research and implementation practice.

5. There has been continuous and increasing adoption of this approach at national and international levels.

6. Key factors helping towards a better return on investment in the domain include: i) it establishes an implementable, incremental route towards adoption and ii) it broadens industry's opportunities for integration with and investment into customer and partner technology stacks.

Without deeper underpinnings of patient and clinical engagement, engineering discipline, and trust, any new policy will likely lead us along the now familiar path: up hill and down dale, towards yet another policy. It is time for the task to be reformulated. In the spirit of Fred Brooks and his concept of "the mythical man month" (tinyurl.com/znl98), one might suggest, controversially, that a tenth of the money being spent today would, over time, if better focused, deliver far greater benefit for health care than has been achieved to date by the expensive systems and programmes currently deployed. Too much money is being spent on not solving the problem. The requirements and form of an open source ecosystem for health information systems and services deserves realistic and informed consideration. It must be soundly grounded in organizations and methods that can be demonstrated to work, and be sustained.

About the Authors

David Ingram has held posts in industry, the National Health Service, and university medical schools. After undergraduate physics at Oxford and several years in the medical engineering industry, he studied computer science and completed doctoral research on the mathematical modelling of biological systems at University College London. He was appointed to the first UK Chair in Medical Informatics in 1989 and participated, as partner and reviewer, of numerous EU and UK Research Council Health Informatics programmes and projects from 1985-2011, including coordinating the EU GEHR Project, which laid the foundations for a standard health record architecture and the openEHR Foundation and community, internationally. In 2011, he established Charing Systems as a spinout company of UCL, to provide services to developers and users of clinical systems, to support their integration within open-source platforms, utilizing the specifications and methods pioneered and made freely available under open license by the Foundation. He is an elected Honorary Member of the Royal College of Physicians, in recognition of his services to medical science.

Seref Arikan has worked in the software industry for 15 years and in the medical informatics domain for 10 years. He is strongly focused on research and development tasks and has wide experience of information technologies and architectures for projects ranging from workflow based systems to national e-health repositories. He has been studying at UCL, pursuing research for a PhD under the supervision of Professor David Ingram, since 2008 and is currently working at Ocean Informatics UK. His research interests are in innovative, high-performance architectures to enable and support computable machine intelligence in healthcare, supported by open source tools and frameworks. He has already released much of this work as open source – the Opereffa framework described here being the most significant item. The clinical context of his PhD programme is the ophthalmology record at Moorfields Eye Hospital in London, where he works alongside Dr Bill Aylward, leader of the openEyes initiative for an open source electronic patient record for eye care.

Citation: Ingram, D. and S. S. Arikan. 2013. The Evolving Role of Open Source Software in Medicine and Health Services. *Technology Innovation Management Review.* January 2013: 32-39.

Sustainability and Governance in Developing Open Source Projects as Processes of In-Becoming

Daniel Curto-Millet

"It's not coordinated, it can't be, because it's all left-field stuff. So GPSoC I knew nothing about. And I mean quite honestly that's the way I would want it to be because I think a thousand blossoms blooming is really the nature where we're at, at the moment, because we are not going to know exactly where the whole thing will resonate and where it will add value. We had no idea that somebody in Cambodia was going to download Opereffa and build a TB national alert system."

An openEHR Board Member

Sustainability is often thought of as a binary state: an open source project is either sustainable or not. In reality, sustainability is much more complex. What makes this project more sustainable than that one? Why should it be assumed in the first place that sustainability is a prolonged state of an ingraced project? The threads are pulled from their yarns in many directions.

This article attempts to reconceptualize some assumed notions of the processes involved in developing open source software. It takes the stance in favour of studying the fluctuant nature of open source and the associated artefacts, not as well-defined objects, but as commons that are continually built upon, evolved, and modified; sometimes in unexpected ways. Further, the governance of these commons is an ongoing process, tightly linked with the way in which these commons are allowed to further develop. This perspective of "in-becoming" is useful in understanding the efforts and processes that need to be provided to sustainably govern the development of open source projects and the advantages for managing requirements derived therein.

Introduction

Studying sustainability in open source software is complex because the practices sometimes contradict what we know from traditional software engineering (Crowston et al., 2011; tinyurl.com/alglebm). There are still few theories and only Elinor Ostrom (tinyurl.com/pcxroc) has left us with something akin to a grand theory of the governance of the commons. By theorizing about this phenomenon using the very tools and paradigms that had previously obscured them, she established the notion of the commons and open source as an accepted and worthy field to study. She did this by giving the field a tradition, a rhetoric, and a history that we could all use to legitimize our own work.

This article attempts to problematize sustainability by contrasting notions of order and structure and by veering away from binary state definitions. In analyzing openEHR (openehr.org), an open source software project focused on electronic health records (EHRs) and related systems, this article emphasizes that the governance of an open source project *is a verb*, and *not a noun*, and that the way in which the processes are governed (or otherwise managed) is integral to the ongoing sustainability of the project. Using Ostrom's concept of commons and the theoretical underpinnings of *becoming* from Deleuze and Guattari (1987; tinyurl.com/awujgdr), it is argued that the principal commons in open source are the very processes that create those commons, which are not static resources, but *in-becoming*. Finally, the concept of sustainability in the governance of open source projects is associated with that of change (Tsoukas and Chia, 2002; tinyurl.com/b7xbfof). Further, problematizing sustainability as chaotic vectors with potentially unexpected developments reveals efforts that otherwise would remain invisible and could be helpful in governing its processes effectively.

Methodology

openEHR is an open source project that defines clinically meaningful concepts and describes in which ways these are to be defined. In so doing, it presents itself as a potential standard for the interoperability of EHRs. openEHR exemplifies the new frontiers explored by open source bridging many different expert communities: academics, patients, clinicians, computer engineers, and politicians. Further, it places itself in the still largely unstable area of EHRs (Black et al., 2011: tinyurl.com/bdogg8r; Hayrinen et al., 2008: tinyurl.com/bz6bqzp; Hovenga and Garde, 2010: tinyurl.com/a2t96rx).

The case study presented here is part of ongoing doctoral research focused on requirements engineering and development processes in open source. The findings are derived from 10 open-ended interviews with core members of the openEHR project. Most core members are also board members; others have built strong reputations through work on satellite projects and prolonged involvement. The overall objective was to understand what core members understand by requirement processes in open source, leading to questions of governance, sustainability of processes and community, sourcing of ideas, control, etc.

Why Open Source Governance Should Be a Verb

Typically, it is understood that a project is open source if its license conforms with criteria set by the Open Source Initiative (OSI; opensource.org). At its foundation, open source is a static, legal definition describing what can be done with the source code (Perens, 1999: tinyurl.com/27jxjg7; Raymond, 2001: tinyurl.com/9fsvzua). Whether this matters at all to the general public, or whether it has any immediate effect beyond the development team, is a problem known as the "Berkeley Conundrum" (Fitzgerald, 2003; tinyurl.com/93g4adm). It matters when considering the reaches that code as law can have, the specific mechanisms of social control it can induce, and the implications to democratic values (Lessig, 2006; tinyurl.com/qyzuhj). Lessig's view is more political, less static. His concern turns from legal code to one of consumption, production, and social responsibility. These issues are even more relevant given the new domains, into which open source has entered and which are far away from its academic and hacker origins (Fitzgerald, 2006: tinyurl.com/9hnxdgv; Lindman and Rajala, 2012: timreview.ca/article/510). When discussing open source and archetypes (abstract representations of meaningful clinical concepts in EHRs), a board member says:

"When I hear open source I tend to think software rather than knowledge so it's quite different. So the philosophy is, the issue is how do you know when an archetype is good. How do you know, the phrase is, how do you quality assure a model? That your fellow colleagues, the developers, that national governments know that this archetype is safe, doesn't contain manifestly bad practice, whatever. And most people take the view that of a kind of waterfall approach, so you get the great and the good and the wise say what the requirements are, some clever people [...] develop the archetypes, and then we pass this off to a standards body, [...] have some experts, blood pressure experts who say oh yes, yes, yes, that's right, they tick the boxes, they will have some formal criteria against which they'll be marking the archetypes and they will pull in experts, maybe cardiologists. Now I just don't think that's going to happen. I don't think it is possible to know **when** *an archetype is good enough."* [emphasis added]

This quotation evidences the new nature of open source software, away from the code, in the knowledge realm; it shows some of the values and goals associated with using an open source approach (quality through edition, diffusion, and acceptance of the archetypes; open source rivalling with standards bodies as an institutionalizing power; and continual and acceptable change. "When is an archetype good enough?" Who can answer that other than someone following an open source process?

Open source, and the artefacts it engenders, are definitions in the making, processes, arguments, and particular engineering models. The knowledge engendered is not a thing, a static good eventually catalogued, it is potentially embedded in a continual process of being made, of evolving. Given that an open source commons is an ongoing construction that can never be considered "finished", it can be difficult to place a commons in time and ask the question: "*When* is an open source commons?" To answer the question, and to understand why it is so difficult to answer, we must study the nature of these commons.

Sustainability and Governance as Processes of In-Becoming

Daniel Curto-Millet

Open Source Commons Through Open Source Collective Actions

Ostrom's work was framed by the economics of resource scarcity. Notably, one of the spin-offs of her framework helped inspire a framework on social-ecological systems (SES) (McGinnis, 2011; tinyurl.com/9r5f849). How can Ostrom's work be useful in a field where what is abundant or scarce is not one of the usual resources that we think of (Anderson, 2009; tinyurl.com/a9mkngf)?

What is an open source commons? According to the static, legal definitions of open source, the code is the foremost of commons. It is the central artefact to which people are contributing. However, focusing only on the source code is limiting, because it does not take into account the entirety of what Ostrom calls the "action situation", where actors interact and evaluate outcomes (Ostrom, 2011; tinyurl.com/akfmd3w). In open source, this is not one physical space, but many interrelated ones (e.g., presence in the code, in the mailing lists, in the documentation, in the IRC channels, in the annual conferences, even in the press). It is useful to see that open source is not just online coding, but that it occurs in a wide variety of different media. The rules and engagements are likely to be different in each media, and the ownership of those different spaces depends on various rules and norms of engagement. Ciborra would probably say that these technologies carry different "necessities of hospitality" (Ciborra, 2004; tinyurl.com/addfljo).

The notion of an open source commons is also a fleeting one, with the increasing range of domains into which open source is entering (Fitzgerald, 2006; tinyurl.com/9hnxdgv). The complexity of what a common is, and therefore, the ownership of those commons is much more complex than it used to be. As an example, openEHR could be said to have several layers of commons. The project's goal is to become a standard in health by defining and creating archetypes that in turn define meaningful clinical concepts. These archetypes are based on a reference model that has become an established standard. The reference model was principally inspired by the efforts of openEHR and other previous projects in which the core members participated. Archetypes are potential clinical requirements for any system that adopts openEHR; they describe clinical concepts such as blood pressure. To define archetypes, the openEHR foundation proposes two editors (tinyurl.com/byksdmk), one from a company with goals closely aligned to the foundation, and another from Linköping University. The editors themselves are based on parsers that understand the Archetype Definition Language (ADL). When archetypes are drafted, they are placed in the Clinical Knowledge Manager (CKM; openehr.org/knowledge/), which is a repository of archetypes where they can be discussed, analyzed, reviewed, approved for publication, translated, etc. On top of that come templates, which are supposed to instantiate the deliberately generalized and generalizable archetypes to particular contexts of use.

Now, all of these are resources in the making. All these layers can have their own licensing, and, maybe more importantly, have their own interrelated *action situation*. How could this complexity be managed without undue reduction and simplification? How should these "crops" be studied? What should an archetype look like? Who decides what it should accomplish? Once again, because of the continual, in-becoming nature of knowledge-commons, we fall back to the true commons in openEHR, and in many other open source projects: processes of creation. The processes involved and the knowledge created are so entangled that it is difficult to distinguish the assemblage of actors from the processes they are driving that not only try to reshape the world, but come to a collective understanding of their own collective actions. Since there is no "when" bounding the creation of knowledge-commons to a specific, well-defined time, the next logical step is to study how these knowledge-commons are created, and what are the processes that sustain them.

The Sustainable Processes: Creating Abundant Commons

Sustainability in open source refers to the project's ability to support itself over time (Chengalur-Smith et al., 2010; tinyurl.com/ckvgafl). It has already been studied, especially through the lens of the community, free-riding, and project size (Lerner et al., 2000: tinyurl.com/a8oygl3; 2006: tinyurl.com/9bp78rm).

Recent efforts have looked at processes instead of static commons. Studies have shown that power relations are important in the process of contributing to the source code (Iivari, 2009: tinyurl.com/a4bsl27; 2010: tinyurl.com/ajheqtw). Also, values, culture, and organizational shifts have been identified as key issues in the adoption of open source into corporate processes (Lindman and Rajala, 2012: timreview.ca/article/510; Shaikh and Cornford, 2009: tinyurl.com/9oyo3hv). Finally, technology has been seen to play a role in the way it enables collaboration in a distributed scale (Laurent and Cleland-Huang, 2009: tinyurl.com/a2vh9kq; Noll, 2010: tinyurl.com/9x8subn; Scacchi, 2009: tinyurl.com/ab6ynwx).

Sustainability and Governance as Processes of In-Becoming
Daniel Curto-Millet

It is difficult to place a taxonomy to the current study of open source precisely because of the evolving understanding of its complexity. Open source is a negotiated concept, and the processes of creating open source software can be competitive and conflictive, and it can disrupt technologies beyond expert walls. It is *becoming* an abstract political machine, shifting itself to accommodate new ideas, pushing for changes (Deleuze and Guattari, 1987; tinyurl.com/afzx9r4). Open source becomes a way to diffuse innovation and to act upon it. When asked about the use of open source in developing software in multiple-expert-domain, an interviewee said:

"Well, you could argue that you don't need open source to build that relationship, but the thing about it is that, if you want to build an ecosystem of clinicians and developers all collaborating around the same software, let's say around the NHS [National Health Service], then it needs to be generally open source, or at least the clinical models need to be open source."

Open source becomes an enabler and an enactor of ecosystems. Through its links to its rooted academic history, to the hacker folklore which is slowly dissipating, to the corporate worlds it is entering, to the legal definitions that impose obligations and grant rewards, and so many other links, it creates a viable alternative to the development of worldly projections. Some would say that it has created itself as an obligatory passage point, an indispensable question that has to be asked when thinking about developing a new software project (Callon et al., 2009; tinyurl.com/8zgbqzc). "Should we go open source?" is implanted in practice, just as the software engineering norm "don't reinvent the wheel" has been impressed into every computer scientist. Another interviewee said:

"And the rigour bit, for me, in the scientific world, most of physics couldn't exist without open source software, because that's the way people, you know, software is extraordinarily complex, unless you've actually got it in your hand and you work with it, you don't really know. And there's so much software around in the world that nobody really knows that... And it gets sold for millions and millions of pounds and then it turns out to be not what people wanted. We really need the practitioners in the field to be much more grounded."

This quotation emphasizes another aspect of open source development processes: it is sustainable through the scientific, rigorous, transparent values that it enacts by the publishing of the artefacts. This definition encapsulates the requirements engineers' philosopher stone: how to build the correct system above building it correctly (van Lamsweerde, 2009: tinyurl.com/a4d3tl3; Letier and van Lamsweerde, 2004: tinyurl.com/8n9bj43). In other words, how can the proper processes be employed that will ensure that a useful system is built? This brings the discussion full-circle back to the governing of open source.

Governing Open Source: Sustainability is a Process

If the processes are so important in defining continuous knowledge-commons that spring out of open source, how should their management be studied? Can they be managed at all? Clearly, knowledge-commons require continuous efforts, but how can their processes be sustainable? Going back to Ostrom's work, the cited interviews lead us to wonder how sustainable processes can be governed in open source. Actors, it is argued, are one of the principal elements.

In institutional theory, a major component of sustainability is the shared meaning given to norms (Ostrom, 2011; tinyurl.com/aqlanst). A shared meaning, even inside open source contexts implies some form of stability. As Schweik and English (2012; tinyurl.com/9tl3myo) put it: "Institutions – social norms and formalized rules along with established mechanisms of rule creation, maintenance, monitoring and enforcement – are the means through which direction control and coordination can occur." This assertion presupposes an establishment of stabilizing forces throughout open source projects. It is worth wondering to what extent these are accepted, if not debated openly. In open source, even basic and fundamental assumptions are put into question, forming part of the learning process (Dueñas et al., 2007: tinyurl.com/aytv5x7; Shaikh and Cornford, 2004: tinyurl.com/bzzzjr4). Also, given the usually informal nature of open source software development, how can the invisible, tacit rules be taken into account (Iivari, 2010; tinyurl.com/ajheqtw)? Are stability and sustainability too strongly associated? Are order and routine erroneously taken as the norm (Tsoukas and Chia, 2002; tinyurl.com/b7xbfof)?

Ostrom's work helps when analyzing institutional norms and dysfunctions in the governance of commons. Through the identification of dysfunctional institutions, we can ponder on the tensions between sustainability and stability in open source. Usually, taken together, dysfunctional institutions give the impression of instability, and open source can be seen as such a disruptive force (Carlo et al., 2010; tinyurl.com/an9gdue). Could instability contribute to maintaining sustainability? If projects become too stable, they could end up losing momentum. Shaikh and Cornford (2004; tinyurl.com/bzzzjr4), found that the learning processes in open source vary depending on the community, technology, code, and even basic, concepts that are questioned but induce collaboration and conflict. Thus, stability is, just as sustainability, a relative concept. This is on par with Ostrom's research, where actors evaluate previous outcomes of an *action situation*, opening the door to much more chaotic and irregular evolutions. When defining actors, Ostrom limits the set to "single individuals or as a group functioning as a corporate actor" (Ostrom, 2011; tinyurl.com/akfmd3w). This is somewhat limiting, but given that the intended audience were economists, as information systems scientists, we can enlarge the population of actors to other entities, agential or not.

In Ostrom's work, actors play an important role in the governance of projects. The actors form an integral part in shaping the processes of creation. What actors contribute to the sustainability of open source processes, and in what degree? This is a difficult question to answer, and will likely depend on the project. But the list is much more varied than it would otherwise seem. Entities understood as economic resources are much more than static objects devoid of action. Some even attribute sentiments to them (Ciborra, 2006; tinyurl.com/chqp8rx). Commons (knowledge or otherwise) are not only resources, they are living entities to which are assigned and which themselves assign centres of gravity and inscribed behaviours. Their properties, their beings are infused with materiality. They are well and alive, and they have an enormous influence, despite being "things". Take another quote from an openEHR board member in a recent project board meeting:

> *"The analogy that comes to mind is the interaction between publishers and librarians. In the context of librarianships, you have national repositories [...] you have some kind of governance framework around the numbering and cataloguing [...] and you have an ecosystem of publishers. You need a new kind of governance which recognises the curation, the librarianship, the skills, is an analogy related to books, there's going to be a correlate of that in the context of archetypes, templates, and there's also going to be a world of publishers."*

The recent debates in the publishing world have provided an analogy to explore and understand how the project itself could or should evolve. It is a new exploration to different types of worlds that appear unexpectedly open. Who could have thought at the beginning of the project that it could learn from the publishing industry?

The knowledge-commons mentioned in this quote (i.e., the archetypes and templates) are thought to be static resources, which, in a matter of seconds, are disembodied, reshaped, and reformed into academic papers, peer-reviewed journals, processes of governance, and curation of books that are seen to merit their emulation. The properties of these knowledge-commons are so flexible in their definition and their processes, that they escape the static view attributed to actor-objects to such an extent that they evade the boundaries between objects and subjects. This is relativism. The knowledge-commons, what they are and what they could be, tear and pull at the project members, influence their view on their governance, and even demand curation.

In this sense, governance is not merely accepted and established institutions, rules, and norms, but also projections of possible worlds, competing values that define the project's essence, historicity, past arguments, motivations, and many other fleeting concepts (Scacchi, 2002; tinyurl.com/bqcwrgn). And hence, the difficulty to understand open source projects only through the evaluation of outcomes, when the worlds that are projected are so difficult to grasp. What IP license should be applied to what artefacts? What effect will the licenses have on the uptake by future community members? These evaluations depend, not only on outcomes, but also on the chaotic projections of possible worlds; they may lead to a positioning of the project in some way or another, yet will remain malleable and subject to change. What might be interesting to study are the efforts to cement those evaluations, to create those fleeting institutions, or as Callon would put it, to objectify them (Callon et al., 2009; tinyurl.com/8zgbqzc).

Thus, the evaluation and exploration is not without consequences. Opening the project to the outside and questioning its internal processes is crucial in the sustainable governing of open source. The actors, intentionally or not, initiate a tentative alignment with possible worlds, embracing uncertainty so as to better cope with it. Does openEHR have the necessary mechanisms to "curate" archetypes? What would a curated archetype look like? What are the processes that need be implemented for allowing the archetypes to fit unknown requirements? Is the project relevant in this new world? A seemingly invisible negotiation takes place to align the project to possibly relevant new actors.

Conclusion

This article tried to problematize the nature of sustainability in open source software. Given the open properties of the knowledge-commons present in open source, their management is much more amenable to changes. Continuous efforts are made to sustain the project and adapt it to the demands of the changing environments, sometimes questioning basic concepts, their purpose and meaning. Sustainability, therefore, is much more than a simple binary state, but the continuous efforts of assemblages of varied actors (technologies, democracy, quality through diffusion and adoption), unexpected alliances (publishing world), collective efforts, and values. These actors are crucial to the sustainable governing of open source. Their efforts and enactions, supported by their values, forces open source projects to explore worlds it had not imagined, in turn questioning its own place and purpose, and the adequateness of its processes.

Sustainability in open source, therefore, is not a one-time buy-in option. It is a continuous process of engagement, of negotiation of even basic concepts. Sustainability depends on the evolution of commons created by project communities. New communities can, sometimes unexpectedly, become integrated into the project, redefining the contexts of use, processes, and purpose. The project, then, is introduced into arenas that were not anticipated. In this article, I have attempted to make apparent that the development of open source projects needs a considerable amount of rethinking in terms of governance, which can only be achieved through the understanding of specific open source concepts: principally, the *in-becoming* nature of commons and underlying processes of development, particular values, fluctuating contexts of use and boundary-less sourcing of requirements.

Acknowledgements

The author would like to acknowledge the support and cooperation he has received from the openEHR board and other members of the project. People approached have been universally accessible and candid in providing information. This openness has been essential in conducting this research. The author also would like to thank the Fundación Ramón Areces for their financial support.

About the Author

Daniel Curto-Millet is a PhD student at the London School of Economics and Political Science. His research focuses on studying requirements engineering and innovation in open source contexts from new perspectives. He has presented his work at a number of international conferences including the Academy of Management conference and the European Conference of Information Systems Doctoral Consortium. He has a background in Software Engineering from University College London and has worked for the DG DIGIT of the European Commission.

Citation: Curto-Millet, D. 2013. Sustainability and Governance in Developing Open Source Projects as Processes of In-Becoming. *Technology Innovation Management Review.* January 2013: 40-45.

Q&A

Matt Asay

Q. Is Open Source Sustainable?

A. In 2008, many thought the open source movement could not survive the widespread adoption of open source software without commensurate contributions back, whether in code or cash. Since that time, however, open source has flourished, and it has become robustly self-sustaining.

This dramatic improvement in the health of the open-source ecosystem derives from two primary trends: a move toward more permissive, Apache-style licensing, coupled with an increase in open-source contributions from web technology companies like Facebook. Driving both trends is an increased emphasis on attracting developer communities, and not simply dollars. Perhaps surprisingly, then, the less the open-source community has focused on financial sustainability and more on developer sustainability, the more money it has made and the more sustainable it has become.

Open Source, Four Years Ago

Open source has the chance of becoming a nonrenewable resource if enterprises consume it without contributing cash or code back. Yes, there will always be open source software that doesn't rely on corporate patronage, ***but it may not be the type and caliber of code that enterprises require.*** (Asay, 2008; tinyurl.com/6opr3p)

When I wrote this back in 2008, I firmly believed that open source was dangerously veering toward an unsustainable state. After all, enterprise adoption of open-source software was booming as the global economy tanked, but the same companies that were happy to use open source were usually not willing to contribute back.

Soon after, however, the industry changed significantly, paving the way for the biggest boom in open source software development in history. The reasons are two-fold: first, open source licensing strategies became much more sophisticated, and second, a new breed of enterprise arose that gleaned significant benefit from giving away open source software without needing anything in return.

Open Source Licensing Grows Up

The early years of open source were marked by fractious religious wars between advocates of free software and open source software. The free source crowd looked to the GNU General Public License (gnu.org/licenses/gpl.html) as the ideal license to enforce user freedom, because it forced iron-clad guarantees that the code in question would remain open source, while the open source group focused on a broader definition of freedom, preferring the more liberal Apache license (apache.org/licenses/). While the GPL took centre stage during the formative years of the free and open source software movement, governing the development of Linux and other significant projects, over time it has given way to a trend toward Apache-style licensing.

The reason? Developers.

Adherents of GPL-style licensing continue to insist that all software *should* be free and the license must manacle any attempts to extend it with proprietary software. Unlike the Apache license, the GPL embeds the decision as to the code’s open source nature into the code itself: if you use the software, you *must* release any derivative works as open source. You have no choice. Apache is very different. Apache adherents believe that software *can* be free and is perhaps best when free. But, these same adherents are not prepared to force other developers to agree with them, and the license does not embed a final decision into the code itself. Downstream users of Apache-licensed software are given wide latitude as to how they use (and license) the software.

As the importance of developers has grown in the software industry, Apache-style licensing has boomed, outpacing GPL-licensed projects to a considerable degree. These trends are illustrated in Figure 1, which is reproduced from Matthew Aslett's (2011; tinyurl.com/7ujq7sj) blog post on this topic.

In many ways, this decline reflects a rejection of the premises underlying free software licensing, with its rigid focus on *software freedom*, in favour of a broader

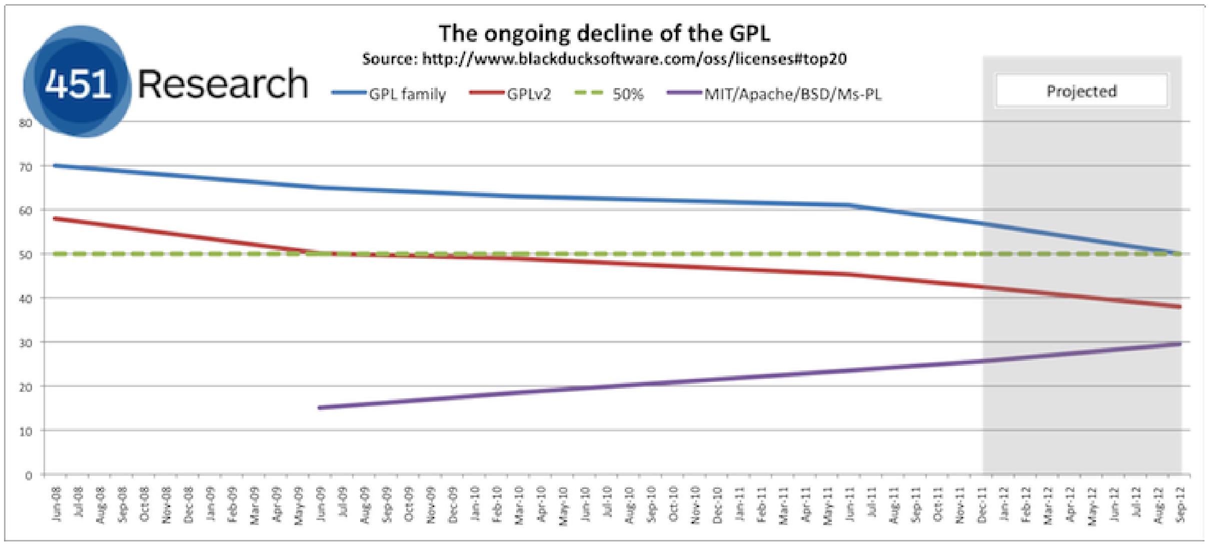


Figure 1. The rise of Apache-style licensing and the decline of the GPL since 2008. Used with permission by 451 Research.

emphasis on *developer freedom*. It is also a rejection of the early open source business model, which used the GPL to essentially build a proprietary software business from free software licensing. That is, with the GPL, an open source vendor could give away its software under a license that many viewed as radioactive, to the extent that it was completely open… and effectively proprietary. Given that the GPL requires any derivative works to also be licensed under the GPL, including third-party software that links to GPL software, depending on how the developer links the software, the GPL puts fear into the heart of legal counsel of would-be users and commercial developers of GPL software. It is free to use, but the possibility of "tainting" one's code is a risk many simply refuse to take. As such, it is open but closed to such companies, that is, impossible for them to accept using without purchasing a proprietary license to use the GPL software.

The seeds of this trend toward permissive licensing were planted in the mid-2000s as legal departments within large enterprises tried to figure out ways to safely embrace open source software. The GPL and its peers nearly always raised red flags, but Apache and its ilk were given a green light. If you were a developer working within Citi Group or Electronic Arts, it was much easier to get a project done with Apache-licensed open source software than GPL-licensed software, because Apache-licensed software makes essentially no demands on users of the software, putting the hearts of legal counsel at ease.

It was not that developers only took their cues from their legal departments. The exigencies of the GPL weighed down development, requiring a degree of license management that was as burdensome to the developer as it was frightening to the attorney. To the mainstream developer without a political axe to grind, Apache offered the path of least resistance to getting work done. This was as true for the solo developer as the corporate developer.

The Web Giants Get Involved

Even as the traditional enterprise grappled with the licensing issues imposed by free and open source software, a new breed of enterprise sidestepped these issues completely. Facebook, Google, Twitter, and other web companies did not distribute software, and so they were able to freely use both Apache-licensed and GPL-licensed software without bothering about contribution requirements. For years, they did just that, scaling out massive infrastructure on open source software, such as Linux and MySQL, that they modified but did not contribute back to.

Not much, anyway.

Facebook changed all this. Facebook's attitude toward open source has always been one of "freely given, freely give," even as Google and Yahoo! kept much of the open source modifications they made to themselves, arguing that few companies besides direct competitors were in a position to use their software effectively. Facebook openly contributed to open source projects such as MySQL and PHP for some time before it started to release its own innovative open source projects such as Cassandra (cassandra.apache.org).

Since Facebook set the tone, Google, Twitter, and others have followed, releasing some of the industry's most promising software, such as Hadoop (hadoop.apache.org), Storm (storm-project.net), and many other projects. Unlike their more traditional cousins in banking, retail, or other industries, these web giants do not really view software as a competitive differentiator, but instead believe that operating this software at scale is what distinguishes them. They also do not have to worry about directly monetizing open source software, so they can release fantastic software with an eye toward developer adoption, not revenue.

This new strategy was a big upgrade over an earlier phase of open source business strategy, which saw venture capitalists funding open source equivalents to BEA Weblogic (tinyurl.com/76a529v) or Siebel's CRM system (tinyurl.com/383xnjh). Although such open source companies did a great deal to move open source forward, proving that it could be useful for a wide array of enterprise-class applications, theirs was an inferior business model compared to what Facebook and its web peers offered. The web giants sold advertising or other services that happened to be powered by open source software running in a remote data centre. They did not have to worry about selling the software, which gave them every incentive to open source their software.

However, these early open source companies have not stood still. Taking their cue from the web companies, open source vendors such as Cloudera (cloudera.com) increasingly contribute heavily to a core open source project and then sell complementary proprietary software or services. This strategy allows them to contribute fully and without conflict to their chosen open source projects, even while making more money. Not surprisingly, since their primary aim is now developer adoption of these core open source projects, and not direct monetization of them, such companies generally turn to Apache-style licensing.

Where Do We Go from Here?

As companies increasingly turn to open source to drive development and not direct revenue, the incentives are mounting for more and better code to be released under permissive open source licenses like the Apache or MIT licenses. This, in turn, will spur more open source development. In many ways, we are entering a golden age for open source, when projects such as MongoDB (mongodb.org), Hadoop (hadoop.apache.org), and Storm (storm-project.net) push the envelope on innovation, rather than following in the footsteps of proprietary software companies.

Even so, while this almost certainly points to years and years of sustainable open source development, it does not yet resolve the continued inefficiency of software development. At least, not enough. As Red Hat CEO Jim Whitehurst has argued:

> *The vast majority of software written today is written in enterprise and not for resale. And the vast majority of that is never actually used. The waste in IT software development is extraordinary.... Ultimately, for open source to provide value to all of our customers worldwide, we need to get our customers not only as users of open source products but truly engaged in open source and taking part in the development community.* (tinyurl.com/bc2dcxy)

So long as enterprises see themselves as islands of productivity rather than communities of developers, I am not sure that this will change. However, what we are seeing is enterprises gravitating toward common pools of development (e.g., Linux and Hadoop). While it is unfortunate that enterprises are not also collaborating on application software such as CRM or ERP systems, perhaps that is the step they will take once the industry more or less standardizes on the same infrastructure.

In the meantime, expect to see accelerating velocity toward permissive licenses as the race to build communities of developers intensifies. This approach is easier and more effective with permissive licenses such as the Apache license. Even in a world where software is run, not distributed, the nagging doubt imposed by the GPL is simply not worth the bother.

In April 2009, Linux founder Linus Torvalds told me, “There is no upside to pushing freeloaders away.” Although he used the GPL as the license to govern Linux, his comment was in response to a question about the desirability of tightening the GPL to block companies such as Google and TiVo from using free and open source software without contributing back. To some, this was freeloading. To him, it was simply how open source works, with value being created by contributions of code *but also merely by the act of running one's code.*

This mentality has blossomed in recent years. It may have started with developers hoping to foster large communities around their projects, but it has since hit overdrive with the large web companies who have nothing to lose and everything to gain from developers adopting, building on, and even "freeloading on" their software.

This is the future of open source: wide open…and more sustainable than ever before.

About the Author

Matt Asay is Vice President of Corporate Strategy at 10gen, the MongoDB company. Previously he was SVP of Business Development at Nodeable, which was acquired in October 2012. He was formerly SVP of Business Development at HTML5 start-up Strobe (now part of Facebook) and Chief Operating Officer of Ubuntu commercial operation Canonical. With more than a decade spent in open source, Asay served as Alfresco's general manager for the Americas and Vice President of Business Development, and he helped put Novell on its open source track. Asay is an emeritus board member of the Open Source Initiative (OSI). His column, Open...and Shut, appears three times a week on *The Register*. You can follow him on Twitter @mjasay.

Citation: Asay, M. 2013. Q&A. Is Open Source Sustainable? *Technology Innovation Management Review*. January 2013: 46-49.

Issue Sponsor

Lead To Win

Do you want to start a new business?

Do you want to grow your existing business?

Lead To Win is a free business-development program to help establish and grow businesses in Canada's Capital Region.

Benefits to company founders:

- Knowledge to establish and grow a successful businesses
- Confidence, encouragement, and motivation to succeed
- Stronger business opportunity quickly
- Foundation to sell to first customers, raise funds, and attract talent
- Access to large and diverse business network

Twitter
Facebook
Linkedin
Eventbrite
Slideshare
YouTube
Flickr

Apply Now

leadtowin.ca

Author Guidelines

These guidelines should assist in the process of translating your expertise into a focused article that adds to the knowledge resources available through the *Technology Innovation Management Review*. Prior to writing an article, we recommend that you contact the Editor to discuss your article topic, the author guidelines, upcoming editorial themes, and the submission process: timreview.ca/contact

Topic

Start by asking yourself:

- Does my research or experience provide any new insights or perspectives?
- Do I often find myself having to explain this topic when I meet people as they are unaware of its relevance?
- Do I believe that I could have saved myself time, money, and frustration if someone had explained to me the issues surrounding this topic?
- Am I constantly correcting misconceptions regarding this topic?
- Am I considered to be an expert in this field? For example, do I present my research or experience at conferences?

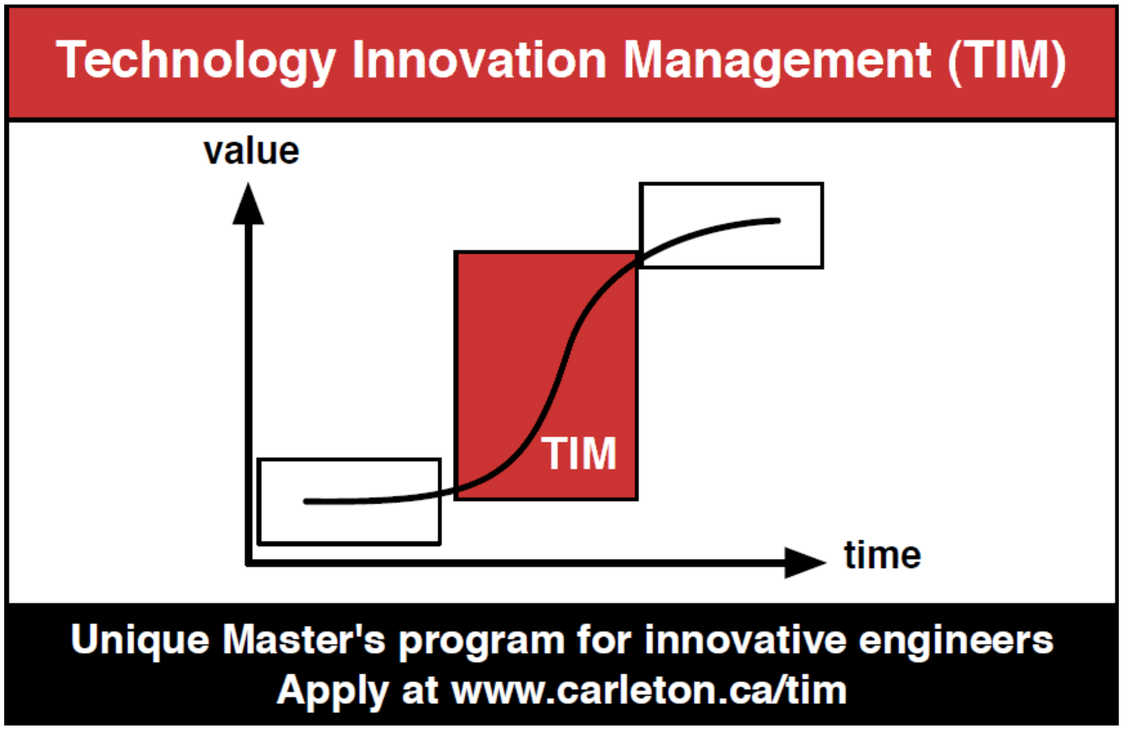
If your answer is "yes" to any of these questions, your topic is likely of interest to readers of the TIM Review.

When writing your article, keep the following points in mind:

- Emphasize the practical application of your insights or research.
- Thoroughly examine the topic; don't leave the reader wishing for more.
- Know your central theme and stick to it.
- Demonstrate your depth of understanding for the topic, and that you have considered its benefits, possible outcomes, and applicability.
- Write in a formal, analytical style. Third-person voice is recommended; first-person voice may also be acceptable depending on the perspective of your article.

Format

1. Use an article template: .doc .odt
2. Indicate if your submission has been previously published elsewhere. This is to ensure that we don't infringe upon another publisher's copyright policy.
3. Do not send articles shorter than 1500 words or longer than 3000 words.
4. Begin with a thought-provoking quotation that matches the spirit of the article. Research the source of your quotation in order to provide proper attribution.
5. Include a 2-3 paragraph abstract that provides the key messages you will be presenting in the article.
6. Only the essential references should be included. The URL to an online reference is preferred; where no online reference exists, include the name of the person and the full title of the article or book containing the referenced text. If the reference is from a personal communication, ensure that you have permission to use the quote and include a comment to that effect.
7. Provide a 2-3 paragraph conclusion that summarizes the article's main points and leaves the reader with the most important messages.
8. Include a 75-150 word biography.
9. If there are any additional texts that would be of interest to readers, include their full title and location URL.
10. Include 5 keywords for the article's metadata to assist search engines in finding your article.
11. Include any figures at the appropriate locations in the article, but also send separate graphic files at maximum resolution available for each figure.


Technology Innovation Management (TIM)
value
TIM
time
Unique Master's program for innovative engineers
Apply at www.carleton.ca/tim

TIM is a unique Master's program for innovative engineers that focuses on creating wealth at the early stages of company or opportunity life cycles. It is offered by Carleton University's Institute for Technology Entrepreneurship and Commercialization. The program provides benefits to aspiring entrepreneurs, employees seeking more senior leadership roles in their companies, and engineers building credentials and expertise for their next career move.

Carleton UNIVERSITY