OpenScrum: Scrum methodology to improve shared understanding in an open-source community

Saptarshi Purkayastha
Department of Computer and Information Science,
Norwegian University of Science and Technology,
Trondheim, Norway
saptarsp@idi.ntnu.no

Abstract: While we continue to see rise in the adoption of agile methods for software development, there has been a call to study the appropriateness of agile methods in open-source and other emerging contexts. This paper examines Scrum methodology adopted by a large, globally distributed team which builds an open-source electronic medical records platform called OpenMRS. The research uses a mixed method approach, by doing quantitative analysis of source-code, issue tracker as well as community activity (IRC logs, Mailing lists, wiki) in pre and post Scrum adoption, covering a period of 4 years. Later we conducted semi-structured interviews with core developers and followed it up with group discussions to discuss the analysis of the quantitative data and get their views on our findings. Since the project is "domain heavy", contributors (developers and implementers) need to have certain health informatics understanding before making significant contributions. This puts knowledge-sharing and "bus factor" as critical points of management for the community. The paper presents ideas about a tailored Scrum methodology that might better suited for open-source communities to improve knowledge-sharing and community participation, instead of just agility

Highlights:

- Quantitative analysis of pre and post Scrum methodology adopted by the OpenMRS project over a period of 4 years.
- Interviews with core developers and focused group discussions with core and community developers to discuss the quantitative analysis and their views on findings of that data.
- Results indicate less agile, but improved shared understanding of design and code-base.
- More active participation in the community and developers feel more community focused.
- A modified scrum methodology is recommended that might be more suited to “domain-heavy” and community-driven open-source projects

Keywords: Agile software development; Scrum; Bus factor; Open-source software; Software Engineering; OpenScrum; OpenMRS;

---

1. INTRODUCTION

Appropriateness of agile methods for emerging contexts (open-source software (OSS), software as a service etc.), ranked first among the top 10 research agenda in the ISR special issue on Flexible and Distributed IS development [1]. Yet, we have seen limited research on agile methods within open-source communities. A recent review by Jalali and Wohlin [2] highlights that Global Software Engineering (GSE) projects with Agile methods are extremely rare. This might be primarily attributed to lack of clear mention that an open-source community is following a certain agile methodology. Some researchers have asked if open-source software development is essentially an agile method [3]. But. Koch [4] mentions similarities, but also points out differences between agile software development (ASD) and OSS development. Thus, until the software development method being used by a community can be evidently clarified to follow one of the fairly well understood agile methods, they cannot be claimed to be the same.

Early work on ASD focused on defining agile methods [5] [6] [7], adoption of agile methods [8] [9], efficiency of agile methods [10] and then more recently focus on empirical studies about post-adoption issues of agile methods [11] [12] and team management [13]. While improved software quality is an observed output, the above researchers highlight "agility" as the most important criteria for adoption of ASD methods. "Agility" in such cases has been used to describe the ability to rapidly and flexibly create and respond to change in the business and technical domains. "Agility" is achieved by having minimal formal processes. Often used concepts to describe "Agility" include nimbleness, quickness, dexterity, suppleness or alertness. These ideas suggest a methodology that promotes maneuverability and speed of response [14].

On the other hand, OSS communities are generally seen as a collaboration of individuals or organizations that participate in software development without contractual bindings, but rather enjoyment-based intrinsic motivation [15]. Some researchers have suggested change in practices (like OSS 2.0) [16] [17], where OSS development is moving towards commercial participation. There is also more recent suggestion that OSS is still largely a combination of commercial ventures and volunteer contributions [18]. Sustainability is often an issue in open-source communities where volunteer contributors "come-and-go" or choose their own tasks [19] [20]. Sustainability of OSS is often described by using the term "truck factor" or "bus factor" i.e. the total number of key developers that would, if incapacitated (e.g., by getting hit by a bus), lead to a major disruption of the project [21]. Another challenge that we see in open-source communities is to gather contributors in projects that are for a vertical domain (health-care, finance, human-resources, etc.) [15]. In many cases, by strategic planning, paid developers will be assigned to work on open source products in vertical domains [16]. If the revenue model for such planning falls short, the developers are moved to other projects.

In this paper, we look at OpenMRS (Open Medical Records System), an open-source electronic medical records platform, which has adopted a tweaked ASD methodology. Since, the project is in a vertical domain of health, it is hard to find skilled volunteers who continue for long periods. Maintaining high bus factor is important for the project's sustainability.

The paper attempts to answer the following research questions:

RQ1. How does adoption of agile methods in OSS change community participation?

RQ2. Can agile methods increase sustainability in OSS communities?

Beyond the above research questions, through the case, we hope that the paper responds to the calls for research of agile methods in OSS development. The case also highlights the challenges and opportunities of switching to Scrum methodology. The case is an avenue for reflection on open-source communities towards metrics of community participation. This will help them understand how their community is at present and what can be done to improve community participation.

The paper is organized as follows. In the next section we look at some of the concepts from software engineering research and agile methodology that have framed the formative and reflexive parts in the research design. In section 3, we describe the mixed method research used for this paper. In section 4, we analyze the tailored use of Scrum sprints in the OpenMRS community and detail out some effects on the project due to use of Scrum. In the discussion section of the paper, we highlight that Agile Methods can be used for knowledge management in open-source projects, instead of focusing on only the agility aspects. Here, we also suggest a tailored Scrum method, we refer to as OpenScrum that might be suited to open-source communities. The last section of the paper concludes by suggesting evolution of agile methods in open-source communities.

2. CONCEPTUAL FRAMEWORK

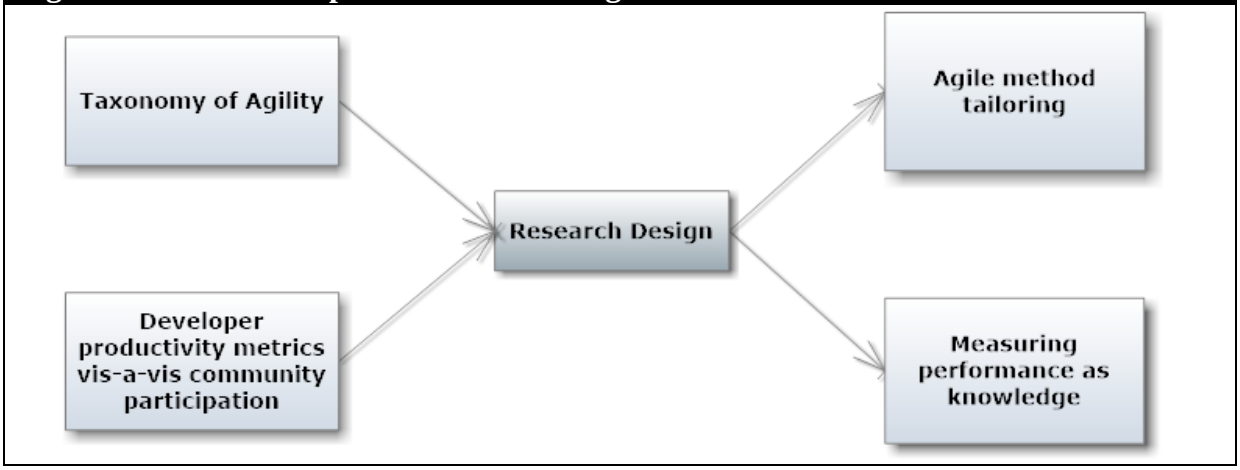
More than a decade ago the Agile Manifesto clarified about the values of agile software development and put forth principles that can be adopted to meet those values. While much of the practices around agile software development have been promoted by practitioners and consultants, there has been a growing need to conceptualize “Agility” [22]. Here, Conboy suggests that Agility comes from two concepts of Flexibility and Leanness. Although used interchangeably, there are conceptual differences between Flexibility and Agility and also between Leanness and Agility. Thus, to be considered agile, the methodology should contribute to creation of change, proaction in advance of change, reaction to change or learning from change. It should also contribute and not detract from perceived economy, perceived quality and perceived simplicity. These allow producing software which is continually ready, with minimum time and cost required to be put into use (ibid.).

While Agility in such terms is an overall measure of the organizational performance to deliver a software product, one should also consider how individual developer productivity is affected by the practice of agile development. While developer productivity has been a hotly debated topic, the 1993 IEEE standard for software productivity metrics defined it as “the ratio of output to the input effort that produced it”. Jones [23] identified 250 factors affecting developer productivity, while more simplistic summary still lists 15 factors [24]. So instead of co-relating multiple factors that affect productivity, it is common to measure output such as in Changes in Lines of Code (CLOC) or Non-Commentary Source Lines (NCSL) [25]. Another measure of developer productivity through interactive participation has been suggested – mainly through the use of code reviews, comments on other people’s code, number of forks, network analysis of contributors [26].

We’ve seen case studies which suggest that communication, co-ordination and control problems in GSE have reduced due to use of agile methods such as Scrum and eXtreme Programming [27]. This and similar research [28] [29] [30] suggests that distributed teams indeed benefit from using agile methods. From all of these cases, we see that there is some level of tweaking done to agile methodology to be relevant to the organization. Tailoring of methods has been observed to play an important role in benefits like reduction of code defects density, delivery ahead of schedule and accurate planning for future projects [31].

While this need for tweaking has been well documented, very little has been written about tweaking agile development to open-source projects. OSS projects might simply be GSE projects in the public domain. Yet, as highlighted in the introduction, sustainability of OSS projects that are managed through community contributions or those that involve multiple stakeholders with different interests, highlight the need for a different kind of tweaking. In all of the above research on GSE, and most research on agile methods [32], we see that management control for changing software development practices could be done by a limited number of stakeholders and all these stakeholders were either organizationally or contractually bound. Even in earlier mentioned GSE literature review [2], "Agile-Open" projects were largely centrally governed. In antithesis to these contexts, consider the Linux project, which in 2011 had 7800 developers for 800 different organizations and 75% of these developers are paid by companies to work on the Linux kernel. Is moving to an agile methodology possible for such a project? What kind of tweaking of agile method will be needed for such a community is an unknown proposition. May be using Linux as a poster-boy for open-source projects is not a useful exercise. But even in small open-source projects which have multiple stakeholders, working with the organizational challenges of adopting agile methods is highly relevant. Figure 1, shows the research design, where we attempt to study agility, as factors of change as described earlier by Conboy and attempt to discuss developer productivity in terms of community participation. The resulting tailored OpenScrum is an output of the research along with measures for community participation.

Figure 1: Use of Concepts in Research Design



The previously introduced concept of "bus-factor" can be understood as a function of knowledge distribution. The more widely distributed knowledge in a community; higher is the "bus-factor". One needs to look at bus-factor also through decision making capacity. If a project has one person making all decisions, the bus-factor is 1 and if this person gets hit by a bus, the project is in jeopardy. Getting hit by a bus, should not be taken literally. It refers to any event that can lead to the unavailability of an individual to the organization. Thus, the processes of KM to increase bus-factor in a software development organization would result in spreading information and decision-making capacity [33]. Infact, at this point in the paper, it is important to mention that the term "Scrum" was coined by Takeuchi & Nonaka [34], which is the agile development method used in the paper's case. Important concepts from their research highlighted, "multi-learning" and organizational transfer of learning. Multi-learning highlights the fact that learning by doing manifests itself along two dimensions – multiple levels (individual, group and corporate) and multiple functions. Knowledge is also transmitted in the organization by converting project activities to standard practice [35]. Thus, OSS project can be understood to follow an agile methodology when individuals as well as the community as a whole, implements agile principles and processes.

3. RESEARCH METHODOLOGY

The research followed a case study methodology [36] to understand the effects of agile methods in its natural context. Case study method is also useful to study post-facto effects, where theory and research are in their formative stages [37]. The research employs a mixed method approach [38] with initially taking an interpretive approach with quantitative methods and later interpretive approach with qualitative methods [39]. Data collection was done from the issue tracking system (JIRA). Individual work units are henceforth referred to as tickets. We analyzed emails from mailing lists (developer [n=18318]; implementer [n=8316], announcement) and source-code; covering the period from Jan 2009 to Jan 2013. Over 3000 tickets were analyzed for factors such assignee, reporter, priority, creation time to resolution time, linkage to source-code and linkage to a sprint or software release. This was done through the use of JQL queries that allow retrieving issues based on selective options from JIRA. Source-code was analyzed in co-relation to the tickets and measured according to the changes in lines of code per developer, number of commits, refactoring of existing code, unit tests and code comments. The research covers code from OpenMRS svn[1] repository for distributed modules, as well as code from git[2] for the OpenMRS core, migration to which happened in August 2012. An Ohloh.net project was created for code analysis by listing various code locations. Additionally, a tool called Fisheye from Atlassian Inc. was used for analyzing activity by developer in terms of code commits and code reviews. Nabble.com was used to get aggregate information about individual contributors on the mailing list. Text mining was not done on the contents of the mailing list, but analysis was done only on the name, email and known organization from the sender's list. Documents on wiki pages which describe design, development and use were analyzed through an interpretive perspective. The wiki is used to collect summary information about discussions and often as a knowledge base about design decisions taken by the community. IRC logs were analyzed for the number of active participants in the IRC, as well as the number of lines of communication was collected to measure the activity in the IRC, similar to that of the mailing list. The mailing list was used to differentiate between developers and implementers. Individuals who have more than 10 emails to the dev list are identified as developers, where as individuals who have more than 5 emails to the impl list are identified as implementers.

This quantitative data was interpreted in relation to the different concepts of agility as presented in the previous sections. This analysis was then shared with each individual core developer through a set of semi-structured interviews which last about 45min to 1hr. A total of 25hrs of interviews were done and 3 group discussions were organized with the core developers. The interviews were transcribed and entered into Nvivo, qualitative data analysis software. Then performed coding based on concepts of "learning", "agility", "knowledge", "release cycle", "participation" and performed thematic synthesis [40]. The resulting themes from the analysis were matched against quantifying words like "more", "less", "increase", "decrease" to verify that the interviewees described the concepts across the interview in the same increasing or decreasing order. Beyond discussing interpretations of quantitative data, opinions were asked on a wide variety of topics such as community participation, developer workload, project management and software development methods in the OpenMRS community. This resulted in deeper understanding of the phenomenon and allowed drawing upon interpretations of core developers. These discussions helped meet the principles of interpretive research [41] such as - principle of contextualization; principle of interaction

[1] Svn or Subversion is a centralized version control system where source code is stored and versioned

[2] Git is a distributed version control system, which allows developers to fork code and work separately on same parts of the source-code.

between researcher and subjects; principle of dialogical reasoning; principle of multiple interpretations - each of which helps bring rigor and validity to the findings.

As in any research approach, case study has its strengths and weaknesses [42] [43]. Case research is important for this type of research, as it allows for study of a large number of variables in a given setting, while these variables do not have to be previously defined [44]. The weakness of such case research is that it is hard to make generalizations or be able to draw conclusions that can be claimed to be valid for all open-source projects. But I take the view that OpenMRS is indeed representative of many similar open-source software communities that work in a vertical domain and have a similar governance and participation model. The OpenMRS governance model is community-driven. Issues are created by community members, weekly developer meetings, weekly implementer meetings, design discussions are on public mailing list or during the weekly meetings. Code review happens in public, voting is used to prioritize features etc. There is a newly formed OpenMRS Foundation with an executive board and community members vote to put a member on the board of directors. Most day-to-day decisions are not taken by the board, but instead through community discussions. The leadership of the OpenMRS community has tried to model itself similar to Mozilla, including having the ex-CEO of Mozilla on the OpenMRS board to get a better understanding of governance principles.

3.1 CONTEXT AND RESEARCHER ROLE

In the paper, I attempt to contextualize ASD in OpenMRS as much as possible. Krutchen [45] highlighted the importance of contextualizing. However, due to length constraints, we don't describe the context of ASD using the full "frog and octopus" model [46], but make maximum attempt to describe all areas, although not as separate sections. OpenMRS is a software platform and a reference application which enables design of a customized medical records system with no programming knowledge (although medical and systems analysis knowledge is required). It has a modular design, where modules are add-ons that extend the functional scope of the system. There are 76 modules installable from the OpenMRS module repository, 125 modules have their source-code in OpenMRS svn. While there are close to 220 OpenMRS modules that are openly available from different sources (github, bitbucket, sourceforge), yet this is only a rough estimate of available modules. Most modules are developed by developers who are not part of the core team. These modules cover broad range of functionality and there is a clear separation of openmrs-core, which has distinct software development lifecycle from modules. While the focus of this research is openmrs-core, we include some modules which are distributed along with the reference application called core & bundled modules. These include FormEntry, HTMLFormEntry, Logic, XForms, DataEntryStatistics, SerializationXStream, Reporting, ReportingCompatibility, HTMLWidgets and PatientFlags. Unless mentioned otherwise, the paper refers to OpenMRS as the "core + distributed modules".

I have been involved in the project as an independent developer for about 6 years, without direct funding from any organization to be part of the software development process. I have also spent a summer internship through Google Inc. at OpenMRS in 2008, through which closer engagement in the community had started. I have been identified as a contributor to the core for many years and have been actively engaged in different roles – as developer, implementer and consultant at for-profit and not-for-profit entities that use the OpenMRS platform. I have participated in many design discussions, roadmap decisions and overall community management discussions before this research. Over the years, I have developed few open-source modules that are used by implementations all over the world as well as

proprietary modules that are used by for-profit and not-for-profit global organizations. All of this highlights that I already had a deep understanding of the community and its practices (implicit and explicit) including roles of core developers and other community members. Walsham [46] classifies such style of involvement as "involved researcher" while doing interpretive research. Yet, the motivation and the decision-making process of changing to agile method, (specifically a customized Scrum method) from a global, distributed software development model were not known to me clearly before this research. This is because the decision was taken by the OpenMRS leadership group and was announced to the community through the developer mailing list. More on the motivation and decision-making process for adoption of agile methodology is covered in the next section.

4. THE EVOLUTION OF SOFTWARE DEV METHODOLOGY IN OPENMRS

OpenMRS is inter-twined software and community, similar to many open-source projects, where the software application development community describes itself as product developers as well as clients of the product. This intertwining is fairly evident in how developers are implementers and implementing organizations contribute developer resources. Yet, some community members are purely implementers who do not have developer resources, while some OpenMRS core developers are allocated only to "*core OpenMRS tasks*" and not to implementation-specific requirements. As we have found out in the study, this is a fairly hard task to balance.

4.1. The globally distributed open-source development model

OpenMRS software development before the move to ASD was like most other community-driven open-source projects. Certain developers are maintainers of specific modules or parts of the system. These developers work for different organizations and have their organizational interests or personal interests. The project started in 2004 as collaboration between two health care organizations and quickly expanded into a global, open-source software community [47]. The project at the time of research had 218 code contributing developers distributed across the globe. Only 41 of these developers are from organizations that implement or support OpenMRS installation. The independent contributors are those who have "come-and-gone" into the project from time to time. The unknown developers are possibly individuals who want an internship position or give a shot at contributing to OpenMRS, but do not actively contribute. In Table 1, we see activity of developer in the community with reference to emails responses, code commits, code reviews. A 30-day period of no-activity makes the developer dormant and continuous activity count is reset.

Table 1: Types of OpenMRS developers and active developers

	No.	Average days active (out of 1460 days)
No. of OpenMRS core developers (C)	13	411 days
No. of organization-backed developers (OD)	41	63 days
No. of intern students (I)	84	71 days
No. of independent contributors (ID)	12	95 days
No. of unknown developers (UD)	68	12 days

This shows that although the core developers are moving forward with development, there is active participation from a larger community of developers, who participate in all types of activities ranging from engaging in discussions to doing code reviews. The role of unknown developers, independent contributors and large number of interns is different and unique to open-source organizations.

In OpenMRS a major percentage of the new developers to the project are student interns, who contribute for a summer or limited period of time. Nearly 75% of the developers fall into category of "community developers" and make up for largest opportunity for the organization to increase commits from these developers from the current 23% levels. This means that without any additional direct cost to the organization, the number of commits can be increased, if the potential of these contributors can be tapped into. On the other hand, OpenMRS has approached the challenge of growth by hiring intern students as Full-time Engineers (FTE), either as core developers or into support organizations. While other open-source projects, mainly run by for-profit organizations have introduced the concept of "bounty", either monetary or "in-kind" [18] to developers who submit code or find specific bugs, OpenMRS has not done anything similar. Thus, the motivation for code submissions to OpenMRS has generally been intrinsic.

As highlighted in the introduction section, volunteer contribution gives rise to a sustainability challenge. OpenMRS also does not make concerted effort to reach out to dormant developers, as highlighted by a core developer. *"We would like to see developers return, but student developers generally remained active only due to Google's funding over the summer and go away to other paying jobs after graduating"*. When asked why, there is no effort to reach out to dormant developers, a general consensus was lack of analytics to know, when and why people drop out of the project. One interviewee mentioned, "*We are experimenting with a CRM system that will be used to track contributors and developers. This will allow us analytics and look at these contributors as leads*".

While analytics and tracking of developers till date has been somewhat easy, OpenMRS leadership realizes that as they've moved to distributed version control and also grown in the number of participating organizations, it continues to become harder to trace code changes across the community. Multiple forks of modules as well as core have been done in recent months by different organizations and developers. But much of the features from these, experimental forks has not returned to the mainline of development. One core developer highlighted, "*We are less concerned about forks. We don't know whether to encourage or discourage forks, when we don't have a mechanism to lure those forks to submit pull requests. We have moved to github and forks are inevitable, but at least easy to track*". While, this suggests the general approach in open-source, that "*free will*" of the partnering organization or contributing developers be respected, it is important to realize that governance and management need active action, instead of passive observation.

In the past, OpenMRS developers have been responsible to maintain and contribute to their own modules. Core developers would generally choose tickets on parts of the system that they are familiar with, and new developers have a set of introductory tickets that have been identified for attempt by new contributors. The developers come from different time zones and are expected to work on the tickets over their estimated time. Before the scrum methodology was adopted in March 2011, developers would generally not have a scheduled meeting, but would hang out in the IRC room. There would be discussions around problems, if a developer had trouble resolving the issue on their own. But there was no formal group communication through which one developer could communicate with all other developers. Also since most developers were themselves the maintainers of the modules, they would rarely get useful responses to resolve an issue. As one developer quoted, "*My module was considered to be black magic that just worked. No one else looked at it and if anything broke, I would be the only person to know where to quickly find it. Instead of showing someone else how to fix it, I'd go and fix it myself*". While most developers were comfortable doing this, if a developer went to vacation or was helping out an implementation, tickets belonging to that

developer's module would be ignored. Bugs remained unfixed and changes took longer to release. Bus-factor in such cases was 1 for many modules and more than 83% of the modules did not receive updates, even though feature requests were made by implementers.

OpenMRS also maintains 2 previous versions, after a new version of core is released. The process of releasing maintenance versions of previous releases is done through the use of what is commonly referred to as backporting of a fix. New features are rarely added to maintenance releases, but sometimes highly voted and relevant features are indeed backported. But in general, maintenance releases only contain bug fixes. While this is useful for implementations to use only the stable functionality, the process of backporting is a time consuming process and there is some lack of clarity in what is allowed to be backported (bug fixes), but sometimes new releases of external libraries, data model changes have also been included in maintenance releases. In Table 2, you can see the release cycles for major version releases. Maintenance releases have not been included for the core, but the bundled modules have their maintenance release distributed and those have been included in the table. Only tickets on which work started and finished between the release period have been counted and long pending or worked on tickets have been ignored. A detailed analysis between of maintenance releases of core and modules is done later in the paper, when we compare the change in performance after adopting Scrum methodology in section 4.3 below.

Table 2: Releases timeframe

Version	Time to release	Tickets resolved = Core + modules	No. of contributors
1.5.0	116 days	177 = 114 + 63	35
1.6.0	202 days	322 = 293 + 29	40
1.7.0	233 days	263 = 167 + 96	50
1.8.0	275 days	320 = 189 + 131	49
1.9.0	352 days	636 = 451 + 185	71

While it takes months before large OpenMRS implementations move to a new release, maintenance releases allow implementations to fix bugs and get important performance benefits. As one implementation-support developer quoted, "*The 1.8.0 release was a paradigm shift in how we were fixing issues to deal with implementations. There were 2 quick maintenance releases made because of performance improvements. Supporting large implementations is about running the right modules, with the right core*". In similar light, a core developer mentioned, "*Implementations differ from each other because of the modules that they use. So, when implementation-support developers commit code, it is to the modules that they use in production. They care less about the core, unless something is breaking a module*".

While the realization that core was getting less relevant for the implementations, it is also understood by the developers that modules needed to be developed at a separate pace from the core. OpenMRS started "Sprinting" using the scrum methodology between the 1.7.1 and 1.8.0 release. The scrum methodology as adopted by OpenMRS community is described in the next section.

4.2. Adopting a tailored Scrum methodology

Scrum is a popular agile software development methodology that focuses on project management in situations where it is difficult to plan ahead; where feedback loops constitute the core element [48]. The core aspect of scrum is the "time-boxed" effort called sprint to complete a set of tasks known as sprint backlog, which have been selected from a larger

product backlog [49]. Instead of discussing general aspects of Scrum methodology, which is well understood and can be read elsewhere, we focus on the use of scrum in OpenMRS. From here on, this is referred to as OpenScrum.

OpenMRS sprints are designed in a way that all participating developers – core, organization-backed as well as community developers work together in sprints of 1 or 2 weeks depending on the module or task at hand. The sprint duration is generally suggested by one developer (who knows the module) or project leader based on their guess of the complexity at hand and from roadmap requirements from implementer meetings. Sprints are also sometimes proposed by developers from the community, implementers or core developers. A sprint schedule is advertised 2 weeks in advance through the developer mailing list and planning starts by nominating or volunteering a Sprint leader. The Sprint leader should be a developer, with adequate knowledge about the module. This developer decides the sprint backlog, by creating new tickets or allocating existing tickets from the product backlog for the sprint. This process is somewhat different from the general role played by Stakeholders and Product Owners in textbook Scrum method. The OpenMRS sprints have the community as the Product Owner and the community as a whole decides what tickets to prioritize by voting on tickets. Although, this has been contentious within the implementer community because core developers have higher chances to influence voting. Also implementations which do not have developer resources are under-represented in such voting schemes. The list of developers who are participating in the Sprint is continuously updated in the sprint schedule. The Sprint leader has to monitor the list of participants so that the tasks that need to be completed in the sprint do not exceed the amount of time that would be required to complete the product backlog. These estimates are fairly complex to make since time estimate of independent developers cannot be accurately made by the Sprint leader. Sometimes, although independent developers self-nominate to participate in Sprints, they do not actively engage and do not spend adequate time during the Sprints.

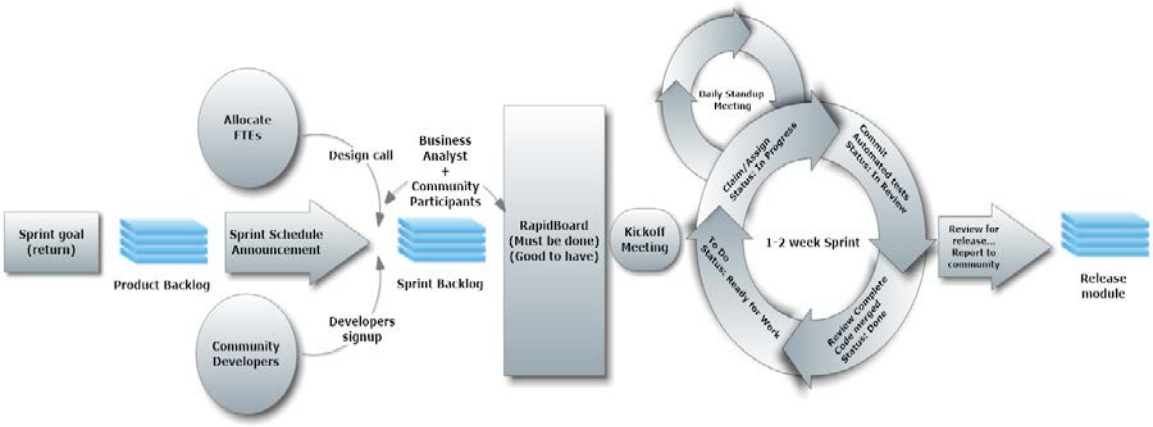


Figure 2: A high-level view of the OpenMRS Scrum

During the planning phase, the design calls play a vital role to translate requirements into working tasks as tickets and to make a guess about what tasks can be completed. OpenMRS involves community members along with a Business Analyst role to complete this process. Implementers are ideally useful for this process, but lack of participation from them, has made the role of Business Analyst to be more important. Community developers signup for the sprint during this period and FTEs from OpenMRS are allocated to the project before or during the design calls. The output of the design call is the creation of a RapidBoard, which

lists out all the activities that need to be completed in the sprint along with their priorities. Figure 3, shows an example RapidBoard, which is representation of the Sprint backlog.

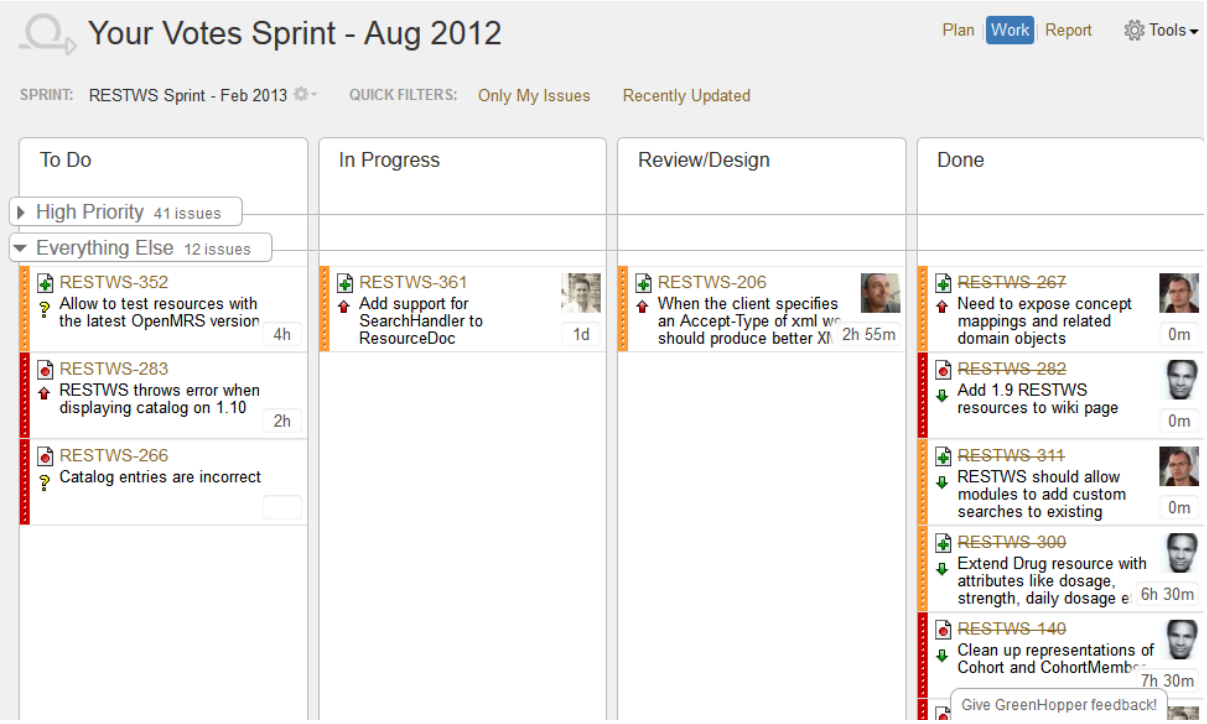


Figure 3: The OpenMRS Scrum RapidBoard

The RapidBoard is essentially an information radiator [14] like a pinup board that is updated automatically and shows the status of a sprint. The OpenMRS sprint starts with a kickoff meeting, which is generally at the IRC. The goals of the sprint, what steps to follow, how to commit code, how to review code, how to merge, what unit tests to write etc. is discussed in this meeting. The prioritized tickets are described in the meeting and community developers introduce themselves to each other during this kickoff meeting. The appropriate wiki pages are highlighted in the meeting; so that new developers can become well versed with some design decisions as well as coding standards to be followed for the sprint.

The OpenMRS sprints also do not have a clear role for a ScrumMaster. A project co-ordinator has recently played the role of the ScrumMaster, yet it is unclear how this role has been used to enforce rules. In textbook Scrum, the ScrumMaster is intended to protect the team from distracting influences, ensure that the rules of sprint are followed and help in gathering resources for sprints. When moving to Scrum methodology, OpenMRS did not have the role of a ScrumMaster. Since July 2012, a ScrumMaster role has been defined, yet activities of the role are vague in OpenMRS. There has been training through OpenMRS University conference calls to explain to the community, the different processes that are followed in OpenMRS scrum. These university calls have had limited participation and there is lack of clarity in the conceptual terms that are part of the followed methodology. In one of the group discussions of community developers highlighted this, "*We would like to have implementers to be ScrumMaster*". Another developer disagreed, "*May be Scrum leader needs to be a developer, but Product Owner should be a single individual instead of the*

community, so that this person can tell us that the module is ready to be released". This shows some conceptual lack of understanding among the developers regarding the scrum methodology in practice and textbook definitions of roles in Scrum.

The OpenMRS implementers mailing list (n=8316) receives about half the amount of traffic compared to the developers list (n=18318). More than 65% of the responses even on the implementers mailing list is from developers. At least from the people who interact openly, we can infer that the OpenMRS community is largely developer driven. This highlights the problem of getting an active role for Stakeholders as well as Product Owner. For instance, when there was an announcement made to organize roadmap meetings which would enable community members to prioritize issues, to create a product backlog that will be used in sprints, after 4 months of attempts to meet, there was no active participation in these calls. Thus, the role of Stakeholders in most sprints has been largely absent. The Product Owner is also different for each sprint, depending on the module being developed in the sprint.

As the sprints continue, there is a daily standup meeting in the IRC, where developers mention what they've been working on and highlight if they have any blockers. Blockers are attempted to be resolved by advice from the core developers soon after the standup meeting. The RapidBoard keeps changing but does not include any time for preparing for release. Merging pull requests, documenting changes, prepare documentation for release are supposed to be done by the Scrum leader, but the OpenMRS scrum model does not have separate time allotted for these kinds of work. The Scrum leader announces at the end of a sprint the results in terms of tickets completed, burndown chart etc. Figure 4 is an example:

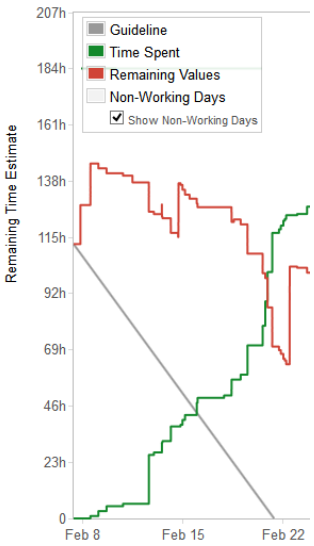


Figure 4: Burndown chart

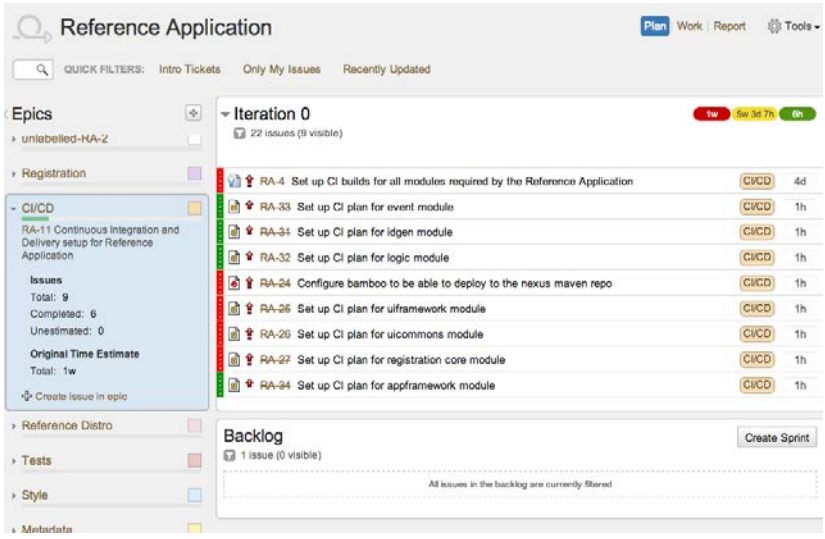


Figure 5: Planning board from Epic to tickets

4.3. Findings and Analysis of the OpenMRS Methodology

While it is evident and somewhat expected that any change in development methodology will result in initial slowdown, we do see that some degree of inefficiency in the way OpenMRS moved to the new methodology. Let us look at the OpenMRS methodology in terms of the Taxonomy of Agility described earlier. It is a combined interpretation of Flexibility and Leanness. Table 3, summarizes the findings after analyzing the parameters.

Table 3: Agility parameters for OpenMRS			
Agility	**Interpretation**	**Leanness**	**Interpretation**
Creation of change ➔ Good increase		Perceived economy ➔ Unclear (NA)	
Proaction before change ➔ Little increase		Perceived quality ➔ Good Increase	
Reaction to change ➔ Moderate increase		Perceived simplicity ➔ Slight decrease	
Learning from change ➔ Good increase			

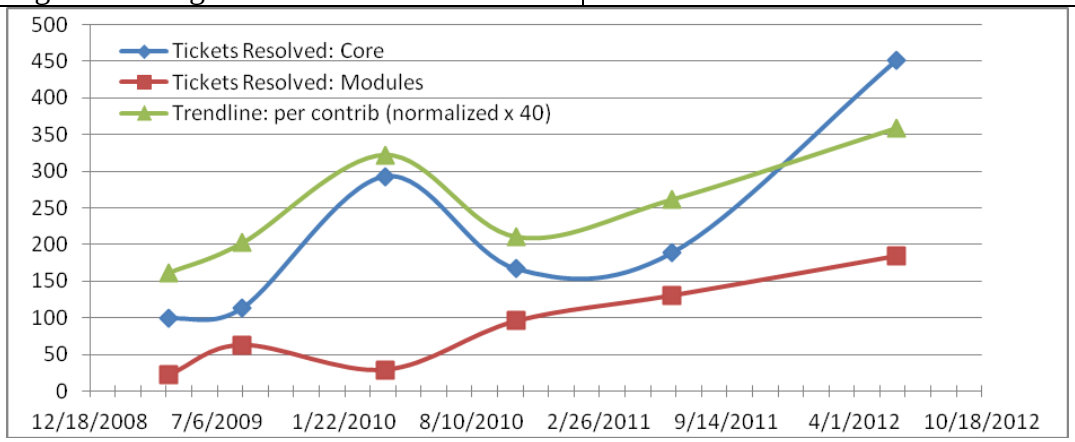


Figure 6: Tickets resolved in each release

Creation of change

Creation of change in a software artifact can be seen from a few different angles. Releases are one way to look at it. We looked earlier that the number of days to release had increased, but that's only the core releases. There was a concerted transition to focus from core to releasing modules quickly. In Figure 6, we see that although there is a clear increase in the tickets resolved over the releases, the number of participating contributors have also increased. Thus, the average contributions (normalized to 40 contributors) in terms of resolving tickets have not really drastically increased. Though it is interesting to observe that after shifting to OpenScrum, there has been stability in the avg contributions from the community. Another interesting fact is that although the focus has indeed shifted to modules, the bundled modules are not the ones that have shown dramatically higher signs of extra work. They seem to be getting the same amount of work done as earlier.

As mentioned in the research design, contributor productivity is another factor of study. Figure 7 shows the CLOC for the top 5 code contributors every month. These are not the same developers, but the top contributors in terms of CLOC to the project for each month. We see that soon after adopting the Scrum methodology the CLOC of the developers went down, but over time the top 5 developers have been committing equal amount of code to the project. This shows that individual developer productivity has become less disparate since adopting OpenScrum.

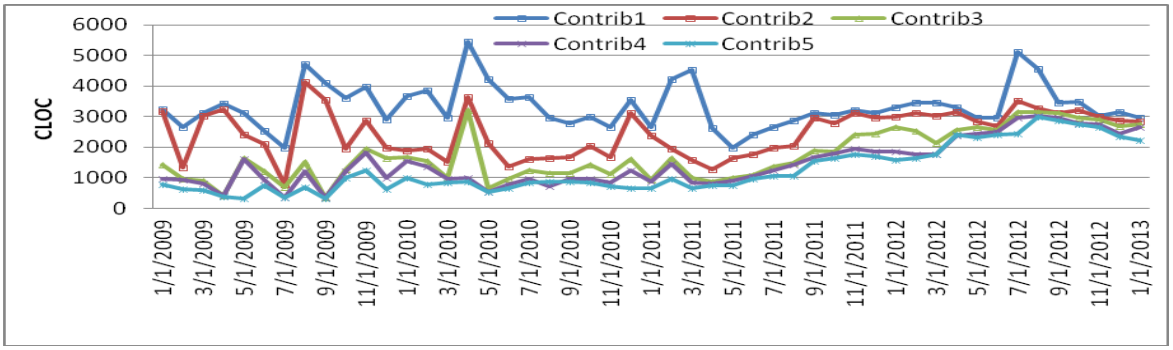


Figure 7: CLOC for top 5 developers

Proaction in advance of change & reaction to change

Lean organizations are often considered to be proactive to change and visionary leaders have the wisdom the see the incoming change in market conditions (Takeuchi & Nonaka, 2011). Out of the total of 48 sprints between Mar 2011 and Jan 2013, there have been 25 sprints that have been on Core (n=19) and Bundled modules (n=6). Others have been on experimental features (referred to as spikes i.e. sprint by 1-2 FTEs) or popular community modules. Yet, as one lead developer commented, "*We wait till something is upon us and then we start experimenting in spikes*". Another developer added, "*We've needed 2 or 3 spikes before anything experimental has been converted to usable modules. Look at RESTWS or OCC*". This adheres to OpenMRS's vision of using the "Story of Floss" – "*Whenever possible, start with the floss* (sic: dental floss)*. See the solution through end-to-end, since this is often the best way to understand the problem and informs the next pass at the solution. In the end, it is rare that we fully understand the problem until the third iteration of the solution. Be agile, open to corrections, and iterate on your solutions. But, most importantly, take action*". From these we might infer that due to sprints and spike-style development, experimentation and chances for innovation have increased. For example, 9 sprints in this period have happened on experimental features that aren't considered to be traditional EMR platform features. These sprints do not directly benefit the large OpenMRS community. Also, decision to run such sprints have not been debated in the OpenMRS community. The leadership of the OpenMRS due to political or with motivations to increase OpenMRS adoption have allocated such sprints. On the other hand, one could also argue that experimental spikes results in lower quality, but better chances of innovation. Quality might be improved in subsequent iterations, if an innovative solution has been found. Thus, we could summarize that we see only moderate to low levels of proaction in advance of change and moderate increase in reaction to change due to adopting this methodology.

Learning from change

There have been 6 sprints on the RESTWS module and each time there was decreasing activity in the number of issues resolved as well as CLOC. This might point to stabilization of code, while moving from experimental spike to quality improvement sprints on which all the devs work together. The number of reported bugs against RESTWS has decreased with each release of the module. Another vital aspect has been that all developers now work on same part of the system and learn similar lessons. As mentioned earlier, developers now learn more and look at varied pieces of code that they would earlier not work on. This includes writing code, as well as reviewing code submitted by other developers. This allows the developers to learn much more and adapt to the different coding styles of developers.

Perceived economy, quality and simplicity

As part of this research, we've not analyzed factors that look at the economy or cost of the development process. During the interviews the managers did not have any kind of cost-estimates before or after the change in the development methodology. Thus no direct or indirect observations could be made on the economic parameter.

The perceived quality of the code has been accepted to be better by the developers as well as the community in general. In 2010, a practice of continuous integration was adopted, which ran unit tests as soon as any code was committed. Along with this, a practice that 2 core developers will need to review code before any ticket gets closed. Although this ensured quality, it was often only the 2 lead developers who would review code. Other developers including community developers would hardly comment or do formal code reviews. With the change to OpenScrum, more community developers are reviewing code and commenting on

each other's code. This is because the developers are watching each other's changes more closely. As one core developer highlights, "*Now missing javadoc comments like @since for newly introduced methods is suddenly more obvious to reviewers. Also writing unit tests is a necessity because it's the reviewer's first comment*".

OpenScrum hasn't necessarily made things simple. Earlier, each developer would fix their own modules, work on parts that they were comfortable with. New modules would be based on requirements from an implementation that would have their developers working on it. The OpenScrum development processes - checks and balances have made things more complex. The GSE method was simple, in the sense that people could work independently, with less effort in co-ordination. The down-side was that with more and more people writing their own modules and with growth of the community without any governance processes, it was becoming harder to ensure quality. So, it is probably worth discussing that for open-source projects although co-ordination is a difficult task, it is worth pursuing, to ensure better quality.

Challenges with Time-boxing and Community Participation

One of the main goals of sprints is that the team focuses on finishing planned set of features in a time-boxed manner. Short cycles emphasize that quick progress needs to be made, even though things can be improved over the next iterations. Another focus of Scrum is that the sprint backlog is completed in a manner that a product is ready to be delivered.

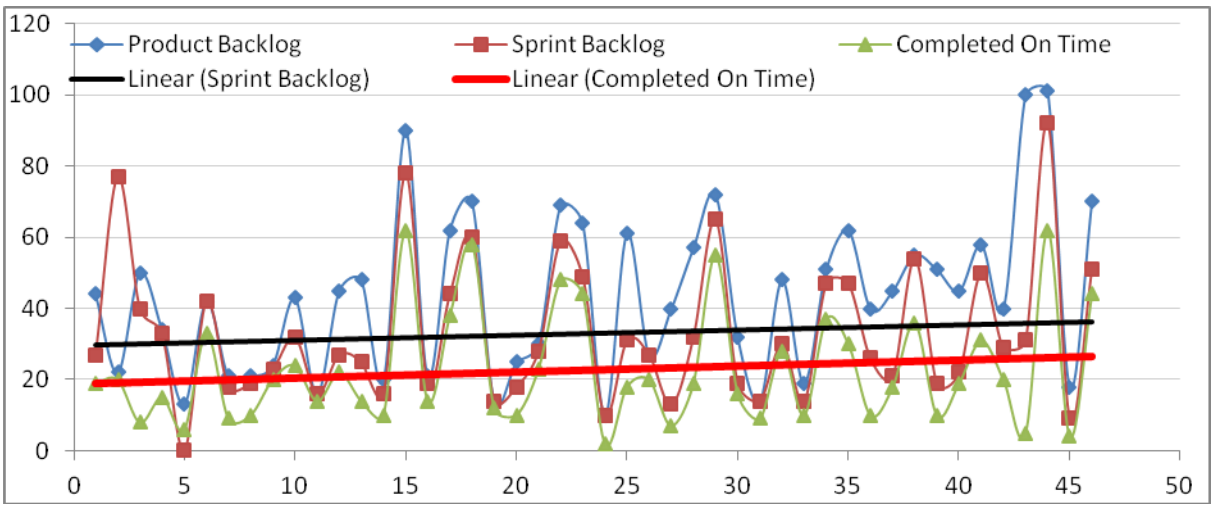


Figure 8: Issues Resolved Vs Planned

From Figure 8, we can see that there has been a constant challenge with the OpenMRS sprints to be able to meet the expected goal in time. Software estimation is generally accepted to be challenging [50], but in open-source communities this estimation becomes nearly impossible because one can never correctly estimate the commitment and time spent by community contributors. The trendline for Sprint backlog Vs Completed on Time shows a constant difference that doesn't seem to have improved over time. Since, not being able to meet the estimated goal with every recurring sprint, the product remains not ready for release most of the time after every sprint. The OpenMRS community in general acknowledges this problem, yet does not have a solution to deal with it.

A clear change since moving to OpenScrum has been increase in community activity. There is much more activity in the IRC, probably due to the daily standup meetings in which the developers communicate about their activities and blockers. There is also a marked increase

in the number of comments that are made on tickets. These comments are made by developers as well as implementers. This shows that people are more actively monitoring and helping each other during the sprints compared to earlier. This is a healthy sign of increased communication in the community and the timing seems to point to the fact that change in the development methodology has resulted in this increased communication in the community.

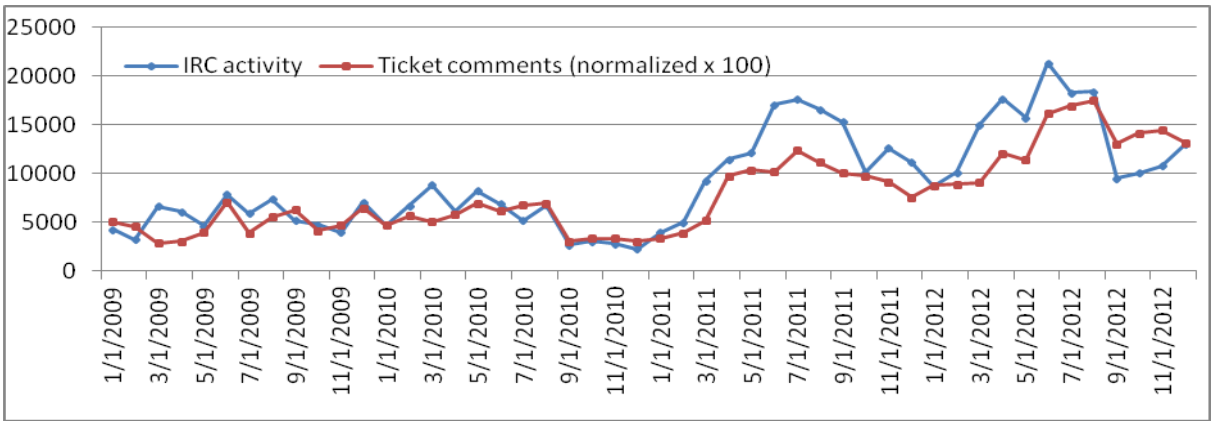


Figure 9: Community activity in IRC and ticket comments

A small note on community participation and priority was highlighted in all the interviews with the developers. While it is clear that sprinting on the most needed aspects is important for the community, it is somewhat difficult to build consensus on what is most needed. For example, requirements for those implementations often get prioritized, that have allocated developer resources. As one core developer put it, *"As we have limited developer resources, supporting implementations of our fellow developers is vital. But sometimes, the loudest voices in the implementer community only get prioritized"*. While the goal of communities should also be to listen to the shrillest voice and provide adequate assistance to them, the interviews highlighted the fact that since moving to OpenScrum, it has become harder to organize sprints that are edge-case requirements. Feature requests that could benefit the community at large or request for new modules, might not get enough votes compared to bug fixes, since large implementations have strength in numbers.

5. DISCUSSION

OpenMRS had a software development process that was used for nearly 6 years before switching to OpenScrum. While it is a continuously improving methodology, some core concepts have evolved in the last 1.5yrs. The paper refers to this tailored Scrum as OpenScrum. Pentaho, the open-source Business analytics suite has attempted a methodology by the same name, but it is not in practice. OpenScrum in essence is ideologically the same basis as the original idea behind Scrum, as coming from Rugby (scrummage). A quick restart of the game (sprint) happens after an infraction. During this restart, a front-row of highly skilled forwards, pushes with itself a team with a common goal. The OpenMRS team is led by such high-skilled core developers and push together with the community developers on a common set of activities during a sprint. Below are some of the differences between Scrum and OpenScrum as observed in practice within the OpenMRS community.

Feature	General Scrum	OpenScrum
Sprint Planning	A product backlog is created and sprint targets are created	Announcement made to the community. Community shows interest and participates in deciding the features that

		need to be built and completed. Design is discussed and available for sprinting
Backlog	A product backlog which is all the wish-list and stories from the stakeholders	A sprint backlog takes focus over a general wishlist. This is a list of available tasks that can be done together by the team. Publicly available for prioritization by community members and contains design documents for coding during the sprint.
Scrum Master	A person responsible for tracking and co-ordinating activities. Helps team to avoid distractions of changing requirements	Scrum Master needs to actively engage community members to participate. Get their views into design calls. Monitor progress of sprints and organize standup meetings. Bring community inputs into a sprint, while not considering them as distractions, but improve deliverables based on these inputs.
Product Owner	A person responsible for defining a Product Backlog and confirming after a sprint that product is ready for delivery.	Much more loose and difficult to define. The community plays the role of the Product Owner by voting on issues and prioritizing tasks.
Information Radiator	A large display board that is shared between the participants in a sprint	Similar to Scrum, but more important as it gives live updates to incoming community members to understand what tasks can be taken up by them, when joining between sprints.
Sprint Retrospective	A meeting during which features developed during sprints is demoed and lessons learnt are shared	A presentation made to the community through videos, community calls and mailing lists. Crediting participants for their activities and sharing burndown charts and timeline of changes to the information radiators
Sprints	A time-bound 4-weeks effort to create an update to a product	Smaller duration with group of developers with different backgrounds working together. Individual developers might participate in “spikes”, which are experimental, but related to ongoing sprints. Developers share experiences in the same meetings as the full sprint team.

In open-source projects, the main objective of using agile methods might not be agility. When developers coming from disparate backgrounds and interest work together, sharing knowledge becomes quintessential. In OpenMRS, certain modules were developed by individual developers and other developers did not know the inner-workings of the module. Many production environments of OpenMRS implementations used these modules and hence it was logical to make these modules bundled with the core OpenMRS distribution. Since only one developer was actively working on the module, it meant that if this developer moved on to do other things or leave the community, there would be no one to maintain the module. Even if a new developer started to maintain the module, it will take a lot of time to

learn about the module after the original developer is gone. Such low bus-factor can only be increased if the community actively spends time and resources to understand how the module works, while the original developer is still with the project. Such processes of shared learning, when creating new modules results in better code review and more people are watching the code that is being written. This should generally improve the code quality, but this research has not looked into aspects of code quality, other than just reported bugs against releases. More substantial research of the code base needs to be done to understand quality improvements due to OpenScrum.

The other goal of using OpenScrum is that best practices which are learnt during a sprint have been actively created by engaging a number of developers. These shared best practices automatically get transformed into organizational practice. This means less ambiguity in the community in reaching consensus about best practices that should be followed. While leanness is a good-to-have outcome from agile processes, it should be fairly obvious that factors such as economic gains or maneuverability are less significant for communities that are working without direct economic bindings. Open-source software development has generally adopted ways of working that are simple for anyone to contribute and leave. This simplicity might be somewhat lost when a community works on building consensus and working together in sprints. Yet, as we have seen, specifically for domain-heavy projects, being able to retain individuals with knowledge is of utmost importance.

6. CONCLUSION AND FUTURE RESEARCH

We conclude by putting forth OpenScrum, a tweaked agile methodology that has empirical basis by which it has helped improve bus-factor in the OpenMRS community. This can be useful to a number of open-source projects that would like to retain developer knowledge and focus on knowledge sharing. To answer the specific research questions – agile methods like OpenScrum have improved community participation and developers know much more about each other's code-base. This in turn answers the second research question of sustainability. More developers continue to know and contribute to more modules. The contributions have widened and bus-factor for a number of modules have increased. We also conclude that Agility might not be an appropriate measure for open-source projects. Instead increasing bus-factor through knowledge sharing, increasing community participation and increasing communication are more important measures in open-source projects.

It will be interesting to use OpenScrum in domain-light (non-vertical sector) open-source projects and see the effect of adopting this methodology in those communities. It will also be useful to view code quality improvements by the use of OpenScrum in open-source projects. OpenScrum is also suited for completely community-oriented projects and might be problematic to use for projects that work in less open fashion or projects that have dual licensing workflows, where organizations need to differentiate proprietary dev team and open-source dev team.

REFERENCES

[1] Ågerfalk PJ, Fitzgerald B, and Slaughter S. Flexible and distributed information systems development: State of the art and research challenges. *Information systems research* 20.3 (2009): 317-328. DOI: 10.1287/isre.1090.0244

[2] Jalali S, Wohlin C. Global software engineering and agile practices: a systematic review. *Journal of Software: Evolution and Process*. 2012 Oct 1;24(6):643–59. DOI: 10.1002/smr.561

[3] Warsta J, Abrahamsson P. Is open source software development essentially an agile method. *Proceedings of the 3rd Workshop on Open Source Software Engineering*. 2003.

[4] Koch S. Agile principles and open source software development: A theoretical and empirical discussion. *Extreme Programming and Agile Processes in Software Engineering*. Springer Berlin Heidelberg, 2004. 85-93. DOI: 10.1007/978-3-540-24853-8_10

[5] Beck K, Beedle M, Bennekum AV, Cockburn A, Cunningham W, Fowler M, Grenning M, et al. *Manifesto for agile software development*. (2001).

[6] Highsmith J, Cockburn A. Agile software development: The business of innovation. *Computer* 34.9 (2001): 120-127. DOI: 10.1109/2.947100

[7] Williams L, Cockburn A. Agile software development: it's about feedback and change, *IEEE Computer* 36 (6) (2003) 39–43. DOI: 10.1109/MC.2003.1204373

[8] Boehm B. "Get ready for agile methods, with care." *Computer* 35.1 (2002): 64-69. DOI: 10.1109/2.976920

[9] Nerur S, Mahapatra R, Mangalaraj G. Challenges of migrating to agile methodologies. *Communications of the ACM* 48.5 (2005): 72-78. DOI: 10.1145/1060710.1060712

[10] Nawrocki J, Wojciechowski A. Experimental evaluation of pair programming. *European Software Control and Metrics (Escom)* (2001): 99-101.

[11] Cao L, Mohan K, Xu P, Ramesh B. A framework for adapting agile development methodologies. *European Journal of Information Systems* 18, no. 4 (2009): 332-343. DOI:10.1057/ejis.2009.26

[12] Mangalaraj G, Mahapatra R, and Nerur S. Acceptance of software process innovations - the case of extreme programming. *European Journal of Information Systems* 18.4 (2009): 344-354. DOI:10.1057/ejis.2009.23

[13] Moe NB, Dingsøyr T, Dybå T. A teamwork model for understanding an agile team: A case study of a Scrum project. *Information and Software Technology* 52.5 (2010): 480-491. DOI: 10.1016/j.infsof.2009.11.004

[14] Cockburn A. *Agile software development: the cooperative game*. Addison-Wesley Professional, 2006.

[15] Lakhani K, Wolf R. Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects (September 2003). MIT Sloan Working Paper No. 4425-03. DOI: 10.2139/ssrn.443040

[16] Fitzgerald B. The transformation of open source software. *MIS Quarterly*. 2006;587–98.

[17] Crowston K, Wei K, Howison J, Wiggins A. Free/Libre Open-source Software Development: What We Know and What We Do Not Know. *ACM Comput Surv*. 2008 Mar;44(2):7:1–7:35. DOI: 10.1145/2089125.2089127

[18] Krishnamurthy S, Ou S, Tripathi AK. Acceptance of monetary rewards in open source software development. *Research Policy*. 2014 May;43(4):632–44. DOI: 10.1016/j.respol.2013.10.007

[19] Mockus A, Fielding RT, Herbsleb JD. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11.3 (2002): 309-346. DOI: 10.1145/567793.567795

[20] Scacchi W. Free/open source software development. *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2007. DOI: 10.1145/1295014.1295019

[21] Stephany F, Mens T, Gîrba T. Maispion: A tool for analysing and visualising open source software developer communities. *Proceedings of the International Workshop on Smalltalk Technologies*. ACM, 2009. DOI: 10.1145/1735935.173594

[22] Conboy K. Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research* 20.3 (2009): 329-354. DOI: 10.1287/isre.1090.0236

[23] Jones C. *Software assessments, benchmarks, and best practices*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[24] Endres A, Rombach D. *Empirical Software and Systems Engineering: Observations, Laws, and Theories*. Addison-Wesley, 2003.

[25] Mockus A. Succession: Measuring transfer of code and developer productivity. *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. IEEE, 2009. DOI: 10.1109/ICSE.2009.5070509

[26] Singh PV. The small-world effect: The influence of macro-level properties of developer collaboration networks on open-source project success. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 20.2 (2010): 6. DOI: 10.1145/1824760.1824763

[27] Holmström H, Fitzgerald B, Ågerfalk P, Conchúir E. Agile practices reduce distance in global software development. *Information Systems Management* 23, no. 3 (2006): 7-18. DOI: 10.1201/1078.10580530/46108.23.3.20060601/93703.2

[28] Paasivaara M, Lassenius C. Could global software development benefit from agile methods?. *Global Software Engineering, 2006. ICGSE'06. International Conference on*. IEEE, 2006. DOI: 10.1109/ICGSE.2006.261222

[29] Herbsleb JD. Global software engineering: The future of socio-technical coordination. *2007 Future of Software Engineering*. IEEE Computer Society, 2007. DOI: 10.1109/FOSE.2007.11

[30] Lee G, DeLone G, Espinosa JA. Ambidextrous coping strategies in globally distributed software development projects. *Communications of the ACM* 49.10 (2006): 35-40. DOI: 10.1145/1164394.1164417

[31] Fitzgerald B, Hartnett G, Conboy K. Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems* 15.2 (2006): 200-213. DOI: 10.1057/palgrave.ejis.3000605

[32] Russo B, Scotto M, Sillitti A, Succi G. *Agile technologies in open source development*. Information Science Reference-Imprint of: IGI Publishing, 2009.

[33] Marchesi M, Mannaro K, Uras S, Locci M. Distributed Scrum in research project management. In *Agile Processes in Software Engineering and Extreme Programming*, pp. 240-244. Springer Berlin Heidelberg, 2007. DOI: 10.1007/978-3-540-73101-6_45

[34] Takeuchi H, Nonaka I. The new new product development game. *Harvard business review* 64.1 (1986): 137-146.

[35] Nonaka I, Takeuchi H. The wise leader. *Harvard Business Review* 89.5 (2011): 58-67.

[36] Yin RK. *Case study research: Design and methods*. Vol. 5. SAGE Publications, Incorporated, 2008.

[37] Eisenhardt KM. Building theories from case study research. *Academy of management review* (1989): 532-550. DOI: 10.2307/258557

[38] Kaplan B, Duchon D. Combining qualitative and quantitative methods in information systems research: a case study. *MIS Quarterly* (1988): 571-586. DOI: 10.2307/249133

[39] Creswell JW, Clark VP. *Designing and conducting mixed methods research*. Thousand Oaks, CA: Sage Publications, 2007.

[40] King N, Cassell C, Symon G. Using templates in the thematic analysis of texts. *Essential guide to qualitative methods in organizational research*. 2004;256–70.

[41] Klein K., Myers M. A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quarterly* (1999): 67-93. DOI: 10.2307/249410

[42] Galliers RD. In search of a paradigm for information systems research. *Research methods in information systems* (1985): 281-297.

[43] Yin RK. Research design issues in using the case study method to study management information systems. *The information systems research challenge: Qualitative research methods* 1 (1989): 1-6.

[44] Cavaye AL. Case study research: a multi-faceted research approach for IS. *Information systems journal* 6.3 (1996): 227-242. DOI: 10.1111/j.1365-2575.1996.tb00015.x

[45] Kruchten P. Contextualizing agile software development. *Journal of Software: Evolution and Process*. 2013 Apr 1;25(4):351–61. DOI: 10.1002/smr.572

[46] Walsham G. Interpretive case studies in IS research: nature and method. *European Journal of information systems* 4.2 (1995): 74-81. DOI: 10.1057/ejis.1995.9

[47] Seebregts CJ, Mamlin BW, Biondich PG, Fraser H, Wolfe BA, Jazayeri D, Allen C, et al. The OpenMRS implementers network. *International journal of medical informatics* 78, no. 11 (2009): 711-720. DOI: 10.1016/j.ijmedinf.2008.09.005

[48] Schwaber, Ken, and Mike Beedle. *Agile software development with Scrum*. Vol. 1. Upper Saddle River: Prentice Hall, 2002.

[49] Schwaber, Ken. *Agile project management with Scrum*. Microsoft Press, 2004.

[50] McConnell, Steve. *Software Estimation: Demystifying the Black Art: Demystifying the Black Art.* Microsoft press, 2009.