An Investigation of Work Practices Used by Companies Making Contributions to Established OSS Projects

Simon Butler
University of Skövde, Skövde, Sweden
simon.butler@his.se

Jonas Gamalielsson
University of Skövde, Skövde, Sweden
jonas.gamalielsson@his.se

Björn Lundell
University of Skövde, Skövde, Sweden
bjorn.lundell@his.se

Per Jonsson
Combitech AB, Linköping, Sweden
per.o.jonsson@combitech.se

Johan Sjöberg
Findwise AB, Göteborg, Sweden
johan.sjoberg@findwise.com

Anders Mattsson
Husqvarna AB, Huskvarna, Sweden
anders.mattsson@husqvarnagroup.com

Niklas Rickö
JAK, Skövde, Sweden
niklas.ricko@jak.se

Tomas Gustavsson
PrimeKey Solutions AB, Stockholm, Sweden
tomas.gustavsson@primekey.com

Jonas Feist
RedBridge AB, Stockholm, Sweden
jonas.feist@redbridge.se

Stefan Landemoo
Saab AB, Linköping, Sweden
stefan.landemoo@saabgroup.com

Erik Lönroth
Scania IT AB, Södertälje, Sweden
erik.lonroth@scania.com

ABSTRACT

Professionals contribute to open source software (OSS) projects as part of their employment. Previous research has addressed motivations of individuals and the ways they engage with OSS projects. However, there is a lack of research which examines and explains work practices used by companies in their engagement with projects. Work practices used by companies to contribute to five established OSS projects are investigated through examination of the actions of employees in public communication channels and draw on our experiences when analysing engagement with the same projects. We find that companies utilise work practices for contributing which are congruent with the circumstances and their capabilities that support their short and long term needs. We find that companies contribute to OSS projects in different ways, such as employing core project developers, making donations, and joining project steering committees in order to advance strategic interests.

CCS CONCEPTS

• **Software and its engineering → Open source model**; *Software evolution*; *Maintaining software*; *Programming teams*;

KEYWORDS

Open source software, company contribution, work practices

ACM Reference Format:
Simon Butler, Jonas Gamalielsson, Björn Lundell, Per Jonsson, Johan Sjöberg, Anders Mattsson, Niklas Rickö, Tomas Gustavsson, Jonas Feist, Stefan Landemoo, and Erik Lönroth. 2018. An Investigation of Work Practices Used by Companies Making Contributions to Established OSS Projects. In *ICSE-SEIP '18: 40th International Conference on Software Engineering: Software Engineering in Practice Track, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3183519.3183531

1 INTRODUCTION

Over the years, professionals in different companies have experienced a variety of reasons for engaging with, and contributing to, open source software (OSS) projects, including improvement of the quality of the software they use and a desire to influence the direction in which the software is developed [6, 18, 26].

When a company and its employees engage with an OSS project that is governed outside the company's control and specific development context it is critical to adhere to established "work practices that are appreciated by community members" [20]. Consequently, any company needs to adopt effective work practices that are accepted and appreciated by community members so that the company can successfully engage with, and contribute to, specific OSS projects. This paper reports on work practices used when companies contribute to established OSS projects.

The collaborative research project LIM-IT [25] investigates the strategic use of OSS by companies. In this study we address the following two overarching research objectives. The first is to understand how companies engage with and contribute to OSS projects governed by foundations and work in combination with other companies to achieve their own and common goals. The second is to understand the motives and decision making behind the approaches used for collaborative work with OSS projects, for example why a particular method of interaction is chosen. In this paper we focus on the first of the overarching research objectives for the study and address how companies contribute to OSS projects by investigating the following research question:

> *How do companies contribute to established OSS projects governed by foundations?*

By *contributing to OSS projects* we mean the submission of feature requests, bug reports, and source code to fix bugs and implement features; as well as contributing to discussions in pull requests, on issue trackers, on mailing lists and in forums, and towards the governance of a project.

We address the research question through an empirical investigation of the public communication channels of a sample of OSS projects. We identify interactions with projects initiated by company contributors and analyse them to identify the work practices used. Our analysis is informed by insights drawn from reflections on the authors' experience of contributing to OSS projects.

This paper makes the following contributions:

- We identify work practices used by companies to contribute to OSS projects.
- We document factors for both the OSS project and contributor that influence company decision making.
- We report insights from industrial praxis into the constraints of the relationships between projects and companies.
- We identify areas where companies might improve their strategies for relationships with OSS projects.

The remainder of this paper is structured as follows. We first present the background (2), the research methodology (3) and details of selected projects (4). Thereafter we present results, which detail interactions between companies and OSS projects (5), followed by an analysis that draws from our experiences (6). Finally, implications for practice and theory are elaborated (7) followed by conclusions (8).

2 BACKGROUND AND RELATED WORK

In a keynote presentation at ICSE 2017, the Executive Director of the Eclipse Foundation explained how many companies strategically engage with OSS projects and claimed that "every software company is an open source company" [32]. Furthermore, practitioners engaged with software deployed from well-known OSS projects (including MySQL and Android) and open source foundations (e.g. Eclipse Foundation and MariaDB Foundation) experience many business benefits, and at the same time a number of challenges that companies need to overcome in order to engage successfully with and contribute to established OSS project [19, 32, 36, 40].

Research shows that many companies in different sectors, utilise software which is developed and maintained in a variety of different OSS projects external to the company [25]. OSS projects typically involve external collaboration with an "unknown workforce" [43]. For these reasons, it is essential for any company to consider the relationship between internal development and how it relates to strategic engagement with OSS projects that may be of strategic importance to their own business. Strategic considerations concerning how a company can maintain control of its own development and engagement with external OSS projects may be critical for long term sustainability that allows for longevity of software [25].

2.1 On Work Practices Utilised by OSS Projects

OSS is, largely, developed in public using systems that track software development and the discussions associated with development. Some discussion takes place on email lists. Much discussion also takes place using *issue tracking systems*, which include Bugzilla[1], and on more recent systems that combine issue tracking with version control systems and software project management to varying degrees. Two of the more commonly used systems are GitHub[2] and Jira[3]. Each of the three systems allows defects with a particular piece of software to be reported. We use *issue* to mean a defect, or perceived defect, in the operation of the software. Contributors may also submit *feature requests* which ask for additional functionality to be added to the software.

Each system also allows the submission of source code alongside both issue reports and feature requests. The code is generally a fix, or suggested fix for the issue, or a suggested implementation of the proposed feature. Contributors reporting issues and making feature requests are not obliged to submit source code. For GitHub hosted projects source code is often submitted in what is referred to as a *pull request*. The pull request is usually the beginning of a process of code review by the receiving project before a decision is made to accept or reject the code. In practice, there is little difference between the process of reviewing submitted source code in the systems mentioned, so we use the term pull request to describe any submitted source code and, where indicated, to a specific submission to a GitHub hosted project.

Some researchers [6, 9] draw a distinction between *core* and *non-core* developers. The distinction is useful, but their definitions vary slightly. For clarity: we define a core developer, or contributor, as someone with *commit* privileges, that is a person who is able to commit source code to the main branches of the project's version control system. All other contributors are non-core contributors.

2.2 Related Work

Questions and answers on mailing lists and in forums, bug reports, feature requests and contributions of source code are made by individuals. Much research has focused on the motivations of individuals who contribute to OSS projects [6] and how they work within projects [9]. There has also been investigations of the coordination mechanisms by which, often geographically dispersed, volunteer teams are able to organise the work of successful software engineering projects [8, 10].

Company involvement in OSS has been investigated to examine their motivations [5], and understand how companies adopt OSS projects to use them in their work [23, 26], to create a revenue stream [11, 24, 26], and as platforms for open innovation [23]. Hauge and Ziemer [24] and Dahlander *et al.* [11] report case studies of companies that create businesses around OSS projects, and the success and failure of attempts to create and grow project communities. Zhou *et al.* [42] investigated the impact of company activity in company-controlled projects on volunteer contributors. The work of Zhou and Mockus [41], and El Asri *et al.* [16] examined how projects can retain contributors, particularly in the first few

[1]https://www.bugzilla.org/
[2]https://github.com/
[3]https://www.atlassian.com/software/jira

months [41]. Singh [38] focused on support requests identifying characteristics of responses and how newcomers to an OSS project community might be alienated by poor quality responses or silence.

Dahlander and Magnusson [11], and Lundell and van der Linden [27] identify tensions between business structures and those of OSS projects. Importantly, as Dahlander and Magnusson [11] observe, the relationship between a company and OSS project is not contractual, which highlights a need for clear governance of projects. Recent work by Germonprez *et al.* argues that the nature of project governance has changed during the last twenty years [22]. Rather than OSS projects being viewed meritocracies, Germonprez *et al.* identify four broad types of governance that include meritocracies [22]. Shaikh and Henfridsson offer an alternative perspective, presenting evidence from a case study of the Linux kernel they observe that governance evolves within an OSS project and also manifests itself as different co-existent forms [37].

Despite the level of attention directed to company and developer interaction with OSS projects, we are unable to identify studies which examine work practices used by interested parties outside the core project team to achieve a company's goals when engaging in and making contributions to a foundation governed OSS project.

3 RESEARCH DESIGN AND METHODOLOGY

In this section we outline the OSS projects investigated and the methodologies used to investigate the actions of software developers and others from commercial organisations who interact with, but do not control, the selected projects.

3.1 Project Selection and Governance

The OSS projects selected for investigation are used extensively by LIM-IT project partners and are of strategic importance for many other organisations. The software created by the five investigated projects are components of, or support the creation of, LIM-IT partners' products or services, regardless of how the solution is deployed (e.g. as proprietary or OSS). Importantly, the selected projects are neither exclusively controlled nor maintained by them, nor are they controlled by another commercial entity, but are maintained by independent foundations or under the aegis of the Apache Software Foundation (ASF) or the Eclipse Foundation. Thus, in this work we investigate involvement in projects where a company has a non-exclusive relationship with the project and must interact with other contributors — commercial entitites and individuals — to achieve its goals. The OSS projects investigated are:

- Bouncy Castle — a cryptographic library
- Leshan — a lightweight machine to machine client and server
- MariaDB — a database management system
- Papyrus — a UML and SysML modelling tool
- Solr — a search engine platform

The project and foundation websites, other online resources, and documentation published with the source code were used to identify the governance systems used by each project.

3.2 Contributing to OSS Projects

To address the research question we examine interactions recorded in publicly available online resources including email lists, online forums, issue/bug trackers and GitHub pull requests to identify the forms of communication available to make contributions, and for evidence of both the conduct of the interaction and its outcomes.

For each of the five projects investigated, we identified the online resources starting with the website and any project pages on foundation websites (see Appendix A for details). Before beginning detailed analysis of the resources, we identified the core developers in each project. Leshan, Papyrus and Solr all publish lists of developers with commit rights. Bouncy Castle and MariaDB do not. Bouncy Castle has very few core developers and they are identifiable from the mailing lists, GitHub and JIRA. The names of MariaDB core developers are not published. However, the more active core developers can be identified by their activity on GitHub and JIRA.

Contributions *initiated* by non-core developers affiliated to companies made to all five projects from the beginning of July 2016 until the end of August 2017 were systematically examined by the first author. The company affiliations of contributors were established from identifying information given in the contributions themselves or associated GitHub and JIRA accounts. LinkedIn was also used to corroborate evidence of corporate affiliation.

Three broad categories reported in the literature were used to identify the wider purpose of the contribution: *bug reports* [33], and *feature requests* [33], and *support messages* or requests [38]. The characteristics of the work practices used by developers to pursue their aims in each contribution were also identified. For example, whether source code was contributed as part of a bug report or feature request. The type of outcome of the work practice was recorded, for example whether the core developers considered the report of a bug to be relevant, and whether the source code was revised as a consequence.

Examples of each work practice identified were analysed and discussed by the authors. Drawing on extensive business experience of interaction with OSS projects, both in senior/managerial positions and as developers, we considered the business and practical aspects of the observed work practices. Relating observations to experience, we also considered what alternative courses of action might have been taken, and whether they might have led to different outcomes. We also reflected on the relationship between the observed work practices and our own, identifying our reasons and motivations for using similar work practices or adopting different approaches.

4 ON THE SELECTED OSS PROJECTS

This section presents characteristics of each OSS project in which work practices are investigated. The selected projects operate under the control of foundations intended to provide a governance framework to help ensure the stability and sustainability of the project. Bouncy Castle is governed by The Legion of the Bouncy Castle Inc., a not-for-profits organisation [4] set up by the project's founders under Australian law. MariaDB is developed with the support of the MariaDB Foundation, which is “non-stock not-for-profit and incorporated in Delaware, USA” [29]. Leshan and Papyrus are projects in the Eclipse ecosystem and are governed by the Eclipse Foundation. Leshan is overseen by the Eclipse Internet of Things Working Group [14]. Papyrus is managed by the Papyrus Industry Consortium [34], a member of the PolarSys working group, which oversees a number of projects focused on model driven engineering and embedded systems [35]. Solr is an ASF project.

Table 1: OSS Project Characteristics

	Control	Licence	OSS Project Established	Foundation Established	Adopted by Foundation
Bouncy Castle	Legion of the Bouncy Castle	MIT	2000	2013	2013
Leshan	Eclipse Foundation	EPL-1.0	2014	2004	2014
MariaDB	MariaDB Foundation	GPL-2.0	1995	2012	2012
Papyrus	Eclipse Foundation	EPL-1.0	2008	2004	2008
Solr	Apache Software Foundation	Apache-2.0	2006	1999	2006

Table 1 provides an overview of the selected OSS projects including details of licensing and when both the projects and foundations were established. The first public release of Bouncy Castle as an OSS project was in 2000 and it has been under the control of the foundation since 2013. Leshan started life as a closed source development project, becoming an OSS project with the Eclipse Foundation in 2014. MariaDB is a fork of MySQL, a long-lived OSS project. Originally a closed source database management system (DBMS), MySQL was established as an OSS project in 1995. MariaDB was forked from MySQL in 2009 and the MariaDB Foundation was created in 2012 to preserve the project's independence. Papyrus is an industry led project to create an independent UML and SysML modelling tool. Papyrus has been an Eclipse Foundation project since before its initial release in 2008 [17, 21]. Solr was proprietary software that became an OSS project governed by the ASF in 2006. The close relationship with the Apache Lucene project led to the integration of the two projects in 2010. Supporting documentation, including mailing lists, changelogs and bug tracking information is mostly available for the entire lifetime of each OSS project.

5 RESULTS

In this section we report on interactions between professionals representing companies and OSS projects. We identify the opportunities for companies to contribute to the OSS projects studied and report observations of how businesses interact with OSS projects when making contributions.

5.1 Project Governance

The governance of an OSS project outlines the management of the project itself and its assets, as well as defining the mechanisms through which the project publishes information, and manages activities and received contributions. Markus' [31] survey of the academic literature identified six categories of governance structures and rules found in OSS projects.

- Ownership of Assets: rules for ownership of intellectual property, foundation structure.
- Chartering the Project: the goals of the project.
- Community Management: rules pertaining to membership and the roles members may have.
- Software Development Processes: rules for requirements gathering, coordination, software changes and release management.
- Conflict Resolution and Rule Changing: rules concerning conflict resolution and changing rules.
- Use of Information and Tools: rules concerning communication, and the use of tools and repositories.

We identify how each of the five selected projects implements governance mechanisms for each category identified by Markus, with the exception of 'Chartering the Project' which all five projects do through their websites. Table 2 presents an overview.

5.1.1 *Ownership of Assets.*

The intellectual property and copyright of Papyrus and Solr, which are Eclipse Foundation and ASF projects respectively, is retained by the foundation in each case. The ASF and Eclipse Foundation contributor licence agreements (CLA) [1, 13] operate similarly whereby the contributor retains the copyright to their contribution while agreeing to give a perpertual licence to the foundation to use the contribution. Leshan uses the same CLA, but copyright is retained by the project's founding company, Sierra Wireless, and the other contributors. The MariaDB Foundation also requires a CLA for code committed using the GPL v2.0 license [30]. However, the MariaDB project also uses a 3-clause BSD licence for contributions and does not require a CLA when that licence is used. Bouncy Castle does not require a CLA. The foundation, The Legion of the Bouncy Castle, holds the copyright of the source code for the Java and C# APIs which are licensed using the permissive MIT licence.

The structure of the ASF is an archetypal meritocratic open source foundation. The ASF is administered by a board and each project is organised, according to the Apache Way [2], by a project management committee (PMC), which elects active members of the community to be *committers*. The MariaDB Foundation supports a similarly meritocratic structure for contributors, though the equivalent of a PMC layer is not present. The Legion of the Bouncy Castle exists to ensure the independence of the Bouncy Castle project from commercial ownership. The direction of software development is informed by developing cryptographic technologies and standards, and is led by both the core developers and the community of non-core developers who have significant domain knowledge. Leshan and Papyrus operate under the rules of the Eclipse Foundation [12]. The technical effort of both projects is led, as is common with Eclipse Foundation projects, by a single company in each case that provides the majority of core developers and undertakes much of the development work. The Papyrus project is managed by the Papyrus Industry Consortium whose members influence the software development strategy through four committees. Leshan is part of the Eclipse IoT Working Group which provides oversight for a number of IoT projects.

Table 2: Governance Characteristics of OSS Projects

	Ownership of Assets	**Community Management**	**Software Development Processes**	**Conflict Resolution and Rule Changing**	**Use of Information and Tools**
Bouncy Castle	The Legion of the Bouncy Castle	*Not explicitly documented*	JIRA	*Not explicitly documented*	Mailing lists, JIRA and GitHub
Leshan	Sierra Wireless and others	Eclipse Community Code of Conduct	GitHub and project mailing list	Foundation Defined	Mailing lists, GitHub and project wiki
MariaDB	MariaDB Foundation	Ubuntu Code of Conduct	Maria Captains mailing list and JIRA	Foundation Defined	Mailing lists, JIRA and GitHub
Papyrus	Eclipse Foundation	Eclipse Community Code of Conduct	Developer mailing list and Bugzilla	Foundation Defined	Mailing lists, Bugzilla, Gerrit and Git, project wiki and forum
Solr	Apache Software Foundation	Apache Way	JIRA	Foundation Defined	Mailing lists, project wiki and JIRA

5.1.2 Chartering the Project. All five projects make clear statements about aim of the project and the software to be created, both on websites and in other documentation. The MariaDB project, for example, states that the software will be binary compatible with MySQL and will always remain OSS, among other things. The Leshan documentation states "Eclipse Leshan is an OMA Lightweight M2M server and client Java implementation." [15]

5.1.3 Community Management. The ASF and the Eclipse Foundation have clear rules on how to contribute and how individuals can acquire status in each meritocracy [3, 12]. MariaDB follows a similar pattern with a core of trusted developers, known as '*captains*', who are able to commit code to the repositories [28]. Captains acquire their status through the development of reputation and recognition within the community, similar to the way Solr and other ASF project developers become committers. Bouncy Castle works in a slightly different way. From observation of the project's JIRA we find that the core developers are members or employees of CryptoWorkshop, a company selling services and support, and the members of The Legion of the Bouncy Castle Inc., the controlling foundation, and accept contributions from a large number of non-core developers.

5.1.4 Software Development Processes. Bouncy Castle, MariaDB and Solr use JIRA as the main tool for coordinating development work amongst the core developers. Bouncy Castle accept contributions from non-core contributors through both the JIRA instance and GitHub pull requests on public mirrors of their private GitHub repositories. The Leshan project uses very little public planning for software development. As the core developers mostly work for one company we conjecture that their geographical co-location makes it easier for them to speak to each other. The Papyrus team use Bugzilla extensively to coordinate their development work.

5.1.5 Conflict Resolution and Rule Changing. The ASF and the Eclipse Foundation provide mechanisms for conflict resolution, and for rule changing. The latter is a foundation wide process, rather than for individual projects. The MariaDB foundation uses the Ubuntu Code of Conduct [7] as community guidelines. However, the precise mechanisms used for conflict resolution, and the opportunities to change them, are unclear from public sources. We were unable to find published documentation detailing any conflict resolution or rule changing processes for the Bouncy Castle project.

5.1.6 Use of Information and Tools. Although the selected projects use a lot of the same tools, as noted in Table 2, they do so in different ways. Leshan uses GitHub to coordinate software development activity, but relies, largely, on the mailing list as a first point of contact for help or support questions, and as a place to discuss significant issues about the project. GitHub is used for reporting issues and for the submission of pull requests. Pull requests can also become a place to discuss the direction of development. Papyrus uses a wider set of tools. Mailing lists and online forums are the first point of contact for those asking about Papyrus, and reporting possible bugs. Bugzilla is used to report bugs and by the core developers to identify and plan other development work. Developers communicate in more detail on code changes using the Gerrit code review tool.

The mailing lists and other communication systems of OSS projects are a repository of project knowledge, recording detail of decisions taken about implementations and the direction of software development. The information is invaluable to help developers and future developers understand the reasons the software is as it is, why particular approaches have been used, and the reasons some have not. All five projects preserve information from the inception of the project, or not long after.

Not all discussions take place on open communication systems and are documented for posterity. Most OSS projects have private communication systems that are reserved for core developers and governance. ASF projects have a private mailing list that is reservered for matters such as the appointment of contributors and the discussion of security issues. Eclipse Foundation projects also have private wikis to record steering committee meetings, for example, and private email lists. Some developers/contributors meet in real life, which is unavoidable when they work for the same companies or attend conferences or meetups. Though they are usually expected to document such meetings in the project infrastructure. However, there has been an increase in the use of closed or semi-closed communications, or '*walled gardens*' [39] where communication is not open, cannot be followed by others involved in the project, and often cannot be archived. An example where a conversation in a walled garden is referred to in public can be found on the MariaDB developers' mailing list "As discussed on slack, lets keep this `Group_bound_tracker` as is."[4]

5.2 Work Practices Used to Contribute to OSS Projects

Governance frames the activities of contributors to OSS projects and the systems through which they contribute to projects. The activities of non-core contributors are concentrated on bug reporting and fixing, the contribution of feature requests, support activities on mailing lists or in forums, and contributions to the management of projects. Through examination of interactions on Bugzilla, GitHub, and Jira instances, and on mailing lists and forums[5] we identified approaches used by non-core contributors when initiating contributions to OSS projects.

5.2.1 Bug Reporting and Fixing. Non-core contributors are a major source of bug reports to a project arising from deployment of the software. Contributors adopt two basic approaches when reporting bugs. One approach is to ask an exploratory question on a mailing list or in a forum. The question typically inquires about some functionality to ensure that the submitter understands how the software is intended to work and whether the observed behaviour is expected, the result of error on their part, or a genuine issue (Example 5.1). Often the contributor will include precise details of the software, hardware and operating system they are using. Where the issue is identified as a bug it may then be reported via the issue tracker by the original contributor or a core developer.

Example 5.1. A user reports that a Papyrus plugin is not loading through a tentative query in the Papyrus forum. The response confirms that there is indeed an issue and the bug is fixed.

A second approach is used where the contributor is more certain they have identified a problem with the software. A bug report is submitted to the mailing list or issue tracker with supporting evidence (Example 5.2).

Example 5.2. A non-core developer reports a key parser bug in Bouncy Castle. The bug report has plenty of supporting evidence. A core developer responds saying the bug has been fixed in the development branch and will be part of the next beta release.

Bug fixes are contributed to projects in response to an identified bug and, sometimes, with a bug report. The mechanism for contributing bug fixes varies according to the tools projects use and their workflow. For example, a bug report accompanied by source code may be submitted as a pull request on GitHub. Most projects prefer that bug fixes are submitted with unit tests that establish the fix works and to support regression testing. Where unit tests have been omitted the core developers will often request unit tests to support fixes as part of the code review process (See Example 5.3).

Example 5.3. A bug report was submitted for Solr by a non-core developer and a patch created with unit tests when requested. The contribution was accepted and pulled into the source code.

Each interaction with an OSS project consumes company resources. Many reports of issues consist of a few steps and messages exchanged between the contributor and the OSS project (as in Example 5.2 above). However, the clarity of the contributor's observations and evidence does not always prevent them from identifying an issue of considerably greater complexity than anticipated, which may require further contribution of time by the non-core developer (See Example 5.4).

Example 5.4. A pull request implementing source code to fix a concurrency bug in interactions between one project and a dependency resulted in architectural changes after a long discussion. The original bug fix, which included revisions made during code review, became unnecessary.

5.2.2 Feature Requests. Non-core contributors also make requests to add new features to OSS projects. As with reporting bugs, the opportunities available for the initial approach include an exploratory question on a mailing list, in a forum or as a GitHub issue, so that the contributor can understand whether the project would be receptive to the proposed feature (Example 5.5).

Example 5.5. A non-core developer asks, in a GitHub issue, about his understanding of functionality he needs to add to Leshan. He checks with the core developers whether he is correct to think that the functionality has not been developed. If so, is anyone working on the feature?

Sometimes, as with bug fixes, non-core developers will submit a feature request with the source code that implements the feature (Example 5.6).

Example 5.6. Functionality that core developers plan to implement in the future is submitted. The contributor needs the functionality now, but implemented it using server code that, while part of the project, is intended for demonstration purposes.

In Example 5.6 a non-core developer contributes functionality implemented for their own needs that the core developers plan to implement much later. The core developers cannot accept the code, because of the implementation was developed against unstable demonstration code, so the pull request is closed. However,

[4] https://lists.launchpad.net/maria-developers/msg10789.html
[5] Appendix A lists the data sources used.

the documentation is updated to indicate the code is intended to demonstrate the project's capabilities.

In some cases, code implemented with a pull request represents a larger feature implemented by the company that is being contributed upstream to the OSS project. A significant amount of software development work is submitted and undergoes review in Example 5.7. On reflection the developer withdraws the pull request and his team submits the source code at a later date as a series of pull requests.

Example 5.7. A pull request implementing significant functionality is submitted by a non-core developer. After code review, the pull request is withdrawn when the contributor decides to break the code down into smaller components and submit those as separate pull requests.

An interesting feature of Example 5.7 is that a commercially unrelated non-core developer comments on the pull request during the code review. The technical point made is minor, and appears to be genuinely helpful. However, comments on source code submitted by non-core contributors are, in many OSS communities, usually only made by the core developers who have to accept the code. This practice is voluntary and Example 5.7 is one of a few occasions we observed where non-core contributors working for a company, and where the core developers are aware of the affiliation, have commented on code submitted by others. The inference is that there is likely to be a strong motivation for an apparently unconventional action. In Example 5.8 a core developer not involved in a code review process intervenes when they recognise a more generic use of a new feature that helps their employer's use of the software.

Example 5.8. A non-core developer submits a new feature. During the code review, a committer, who is not part of the review process, but has a related use case in his business asks for a more generic solution. The code is revised.

Contributions of code may also lead to the inadvertent identification of a more complex problem that results in a lengthy discussion, and the proposal and implementation of fixes that were, perhaps, not anticipated by the original contributor. An example can be found in the Leshan project where a pull request submitting code to fix a race condition between Leshan and a dependency eventually leads to a minor revision of Leshan's architecture that resolves the underlying problem (Example 5.4).

5.2.3 Support. Example 5.6 illustrates that sometimes misunderstandings arise from project documentation. Either the documentation is incomplete, misunderstood, not read, or possibly out of date. Non-core contributors also provide help and support on mailing lists and in forums as in Example 5.9. As users they have detailed knowledge of the deployment and use of the software, and an interest in increasing its user base.

Example 5.9. A non-core developer asks a question about the specific constants to use in a protocol call. Another non-core developer responds giving the answer.

In the following sections we identify, through the reflections of experienced practitioners, why companies adopt particular approaches to achieve their goals and the implications for practice.

6 OBSERVATIONS AND ANALYSIS

6.1 Observations on Work Practices

6.1.1 Bug Reporting and Fixing. Using an exploratory question when reporting a possible bug, as in Example 5.1, has a number of potential advantages. Firstly the report is non-confrontational, so encouraging a response. Secondly, the reporter/questioner does not publicly declare their possible ignorance or misunderstanding to be a bug, and consequently does not diminish their own standing in the project community — something to be avoided. And, thirdly, the question may be one that other users, particularly those new to the software, may also need answered. The submission of bug reports with clear evidence, e.g. Example 5.2, reflect the reporter's certainty of having identified a problem in the software and delegate the implementation of the fix to the core developers.

Example 5.4 illustrates a risk to contributors of submitting bug fixes. The contributor is drawn into a long discussion and additional work because the proposed fix uncovers a larger problem, that the core developers don't immediately identify a solution to. Though the problem is eventually resolved by the core developers, the return on the investment by the contributor's employer is unclear.

6.1.2 Feature Requests. The submission of source code with feature requests can also be a risk for the contributor. The core developers cannot accept the code submitted in Example 5.6, because the implementation was developed against unstable demonstration code. The consequence of the code being rejected is twofold. Firstly the source code represents an investment of business resources. And, secondly, the implementor now has to maintain the code, rather than the project community developing the code further and maintaining it.

The contributor in Example 5.6 appears to have misunderstood the project development plan, and the contributor in Example 5.4 may have lacked sufficient implementation knowledge to create a full solution to the problem. Far wiser, in our experience, when submitting code is the cautious approach used in Example 5.5 that establishes whether the code might be welcome. However, without detailed knowledge of the implementation and any development plans it can be wiser, in terms of company resources, to leave the work of implementation to the core developers.

6.1.3 Support. As noted above, the use of tentative questions to ask about possible bugs provides an opportunity for those active in the community to share their detailed knowledge. While it might seem trivial to advise new users and solve deployment problems, an active and supportive community attracts new users and contributors to the project thereby increasing its sustainability and likely longevity. Furthermore, support contributions can also be seen as a form of bug-triage where the careful, tentative question, identified above, is responded to by a non-core contributor who recognises the usage scenario and can suggest a solution, without a core developer having to address the issue.

6.1.4 The Direction of Software Development. The submission of feature requests is one way to influence the direction of software development. Depending on the governance structure of a project, there may be other opportunities for non-core developers to guide long-term software development. Of the five projects investigated (see Table 1) the Eclipse Foundation governed projects have membership schemes through which companies are able to support the project financially and obtain a place on a committee that provides strategic oversight and leadership for the project.

6.1.5 Project Lifespan. Underscoring our observations is support for the longevity and sustainability of the projects we work with. In one domain, for example, the minimum period we expect to use software adopted from an OSS project is typically between five and seven years. In other domains timeframes can be even longer. In some domains companies have contributed to individual OSS projects for longer, and continue to do so. We also have experience of contributing to the governance of projects, particularly to be part of decisions made about the direction of software development in the long term to align development with our strategic interests.

6.2 Analysis

We find the work practices used in our interaction with the OSS projects are influenced by three main factors: the maturity of the software implementation and the domain; where knowledge and expertise lie; and how the software is deployed within the company.

In our experience, a project where the software implementation is relatively immature and the core functionality under development, such as Leshan, requires greater investment of company resources in the project (in collaboration with competitors or not) to ensure the software meets the company's requirements. Opportunities for a company to work with the community include company developers joining the community, and the company employing core developers within the community. The latter is only possible where the implementation aims of the company do not conflict substantially with those of the community. An example might be the development of new software for an existing standard where the domain is relatively mature and clearly defined.

In other projects, for example Solr, the software implementation is relatively mature and the domain is mostly stable. Development activity is consequently constrained largely to refining the software by correcting errors and responding to changes in external technology, such as the hardware configurations deployed in industry in the case of Solr. In many similar cases the OSS project self-regulates and, because their requirements are already met, our experience is that the involvement of companies using the software is limited to contributing bug reports.

Knowledge or expertise is also a significant factor when deciding to make a contribution to an OSS project. Three broad categories of knowledge emerged from analysis of OSS projects: knowledge of the application domain, knowledge of how the software is implemented, and knowledge of software deployment and use.

First, the application domain knowledge in the software is often the asset that companies exploit to deliver a service or in a product. In some sectors, for example security, or during product innovation there can also be a significant amount of domain knowledge within the company using the software. In the case of Bouncy Castle, for example, the domain expertise and awareness of the community helps to identify new areas for development, as well as to report bugs clearly to the developers.

Second, detailed knowledge of the software implementation is, in our experience, usually limited to the core developers within a project. While we find that it is generally possible to delegate responsibility for implementing bug fixes and feature requests to the upstream project, how the software is deployed by the company can require timely implementation of fixes and features. Where an OSS project is deployed as part of a product, or delivers a critical service, we have found the need to implement bug fixes ourselves. A key challenge, therefore, is to acquire sufficient implementation knowledge to be able to implement meaningful changes. Furthermore, there is a trade-off between implementing a bug fix that resolves the problem until it is fixed in the next release, and a solution that meets the requirements and development plans of the core developers sufficiently that the changes will be incorporated into the upstream project. Striking the right balance is important for the company. Our approaches to the acquisition of implementation expertise are a matter of choice. Employees can acquire knowledge sufficient to implement fixes, though there may be limitations to the level of expertise that can be acquired alongside their day-to-day work. Alternatively, we have hired expertise already within the upstream project in the form of core developers, or developed working relationships with businesses that employ core developers. A similar need for the timely implementation of bug fixes and features in the upstream project also arises during innovation processes where the upstream project provides an API or server that incompletely implements a standard.

Third, deployment and usage expertise lies with the user community, including companies. Contributing that knowledge to the project community makes the project more attractive to new users and contributors.

7 IMPLICATIONS FOR PRACTICE

Drawing on the analysis of work practices used in the five OSS projects, this section elaborates on implications for practice. Working with an OSS project requires awareness of a range of factors to support decisions about what to contribute to the project and how to make the contribution. The factors include an understanding of why the software is used and the role it plays in the company's business, as well as the limitations to the company's capacity to contribute to and influence the project.

An understanding of where *competence* lies in the relationship with an OSS project can help a company make wise decisions about how to interact with the project. For example, without detailed knowledge of the software implementation, submitting anything other than relatively simple code is likely to be a challenging, resource-consuming and potentially unproductive process.

Companies need to identify mechanisms for resolving software problems in a *timely* manner. Where OSS software is a component of a product, provides a business service, or is part of a fast-paced innovation project, the timeliness of bug fixes can be critical. Companies reliant on an OSS component may need to adopt a 'fix now' strategy; and there is more than one way to achieve this goal.

A company also needs to understand how and with whom in a project to form *productive relationships*. Whilst OSS project community engagement and reputation are known to be valuable, relationships forged out of public view may be more useful.

Through an understanding of why it uses an OSS project a company can *focus* on maintaining its own interest. Areas of engagement with a project should be selected carefully so that the activity serves company interests and does not waste resources.

Long running, active OSS projects have a tendency to be self-regulating. The software has clearly defined functionality that is unlikely to change direction overnight. Companies should try to understand how the governance of a project works and identify the amount of vigilance and effort required to preserve their interests.

It is in a company's interests to *promote* the projects they use. OSS projects need publicity to increase the number of users and active participants that improve the prospects of long term stability, thereby reducing the risk to companies using the project.

Foundation structures provide different opportunities for company involvement in the *governance* of a project, particularly in relation to helping to plan the direction of software development. Companies should be aware of the opportunities available to gain strategic influence and use them if appropriate.

The structure of a company and that of the project are also factors that need to be considered when interacting with OSS projects. Previous research reports that "for large companies, it may not be so easy to change established traditions and current work practices. Consequently, adoption of new principles and practices for software development certainly imposes new challenges." [27]. Others have identified variations in the workings of governance, even within a single project [22, 37], which adds further complexity to building a relationship. And we know from our own experience that we have adopted approaches to interaction tailored to our company structure and that of the OSS project. Deeper understanding of the factors behind the pragmatic solutions found by companies will support what Milinkovich identifies as the inevitability of company use of OSS [32].

8 CONCLUSIONS

We have reported on the interactions of companies in the primary and secondary software sectors engaged with five widely used open source software (OSS) projects governed by foundations. The work practices used by companies to contribute to OSS projects are identified and characterised. Through reflections on practice we have identified the variety of strategies adopted by the companies to work with OSS projects crucial to their products and services.

A complex picture emerges of the manner in which companies contribute to OSS projects, despite the outward similarity of the project structures, available communications channels, and business priorities. We found that key factors that determine how a company interacts with an OSS project include the maturity of the software created by the project, the context within which the company uses the software, and the balance of areas of competence or expertise between the company and the project. In addition, companies have a strong interest in the longevity and sustainability of projects they contribute to.

The paper presents rich characterisations of work practices used by companies in five widely deployed OSS projects and illuminates how issues raised by contributors outside the core developers can be acted on. We find several examples in the investigated projects of how issues raised by non core developers from companies have made effective contributions.

In conclusion, we find that companies can adopt effective work practices for engaging with, and contributing to, OSS projects which are of strategic importance for their own business.

ACKNOWLEDGMENTS

This research has been financially supported by the Swedish Knowledge Foundation (KK-stiftelsen) and participating partner organisations in the LIM-IT project. The authors are grateful for the stimulating collaboration and support from colleagues and partner organisations.

REFERENCES

[1] Apache Software Foundation. 2004. Contributor Licence Agreements. Online. (2004). http://www.apache.org/licenses/#clas Last visited: 2017-08-15.

[2] Apache Software Foundation. 2017. Code of Conduct. Online. (2017). http://www.apache.org/foundation/policies/conduct.html Last visited: 2017-09-08.

[3] Apache Software Foundation. 2017. What is the Apache Software Foundation? Online. (2017). http://apache.org/foundation/how-it-works.html Last visited: 2017-08-17.

[4] Australian Charities and Not For Profits Commission. 2017. The Legion of the Bouncy Castle: Registration Details. Online. (2017). https://www.acnc.gov.au/RN52B75Q?ID=4311D441-9D67-4F72-BBDA-16EA9531F478&noleft=1 Last visited: 2017-08-14.

[5] Andrea Bonaccorsi, Dario Lorenzi, Monica Merito, and Cristina Rossi. 2007. Business Firms' Engagement in Community Projects. Empirical Evidence and Further Developments of the Research. In *Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS '07)*. IEEE Computer Society, Washington, DC, USA, 13–17. https://doi.org/10.1109/FLOSS.2007.3

[6] Andrea Bonaccorsi and Cristina Rossi. 2006. Comparing Motivations of Individual Programmers and Firms to Take Part in the Open Source Movement: From Community to Business. *Knowledge, Technology & Policy* 18, 4 (Dec. 2006), 40–64. https://doi.org/10.1007/s12130-006-1003-9

[7] Canonical Ltd. 2017. Ubuntu Code of Conduct v2.0. Online. (2017). https://www.ubuntu.com/about/about-ubuntu/conduct Last visited: 2017-08-17.

[8] Kevin Crowston and Barbara Scozzi. 2002. Open Source Software Projects as Virtual Organisations: Competency Rallying for Software Development. *IEE Proceedings - Software* 149, 1 (Feb. 2002), 3–17. https://doi.org/10.1049/ip-sen:20020197

[9] Kevin Crowston and Barbara Scozzi. 2008. Bug Fixing Practices Within Free/Libre Open Source Software Development Teams. *Journal of Database Management* 19, 2 (2008), 1–30.

[10] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. 2012. Free/Libre Open Source Software Development: What We Know and What We Do Not Know. *Comput. Surveys* 44, 2, Article 7 (March 2012), 35 pages. https://doi.org/10.1145/2089125.2089127

[11] Linus Dahlander and Mats Magnusson. 2008. How do Firms Make Use of Open Source Communities? *Long Range Planning* 41, 6 (2008), 629–649. Issue 6. https://doi.org/10.1016/j.lrp.2008.09.003

[12] Eclipse Foundation. 2015. Eclipse Development Process 2015. Online. (2015). http://www.eclipse.org/projects/dev_process/development_process.php Last visited: 2017-08-17.

[13] Eclipse Foundation. 2016. Eclipse Contributor Agreement. Online. (Aug. 2016). http://www.eclipse.org/legal/ECA.php Last visited: 2017-08-15.

[14] Eclipse IoT Working Group. 2017. Open Source for IoT. Online. (2017). https://iot.eclipse.org/ Last visited: 2017-08-11.

[15] Eclipse Leshan. 2017. Leshan. Online. (2017). https://github.com/eclipse/leshan/blob/master/README.md Last visited: 2017-08-11.

[16] Ikram El Asri, Noureddine Kerzazi, Lamia Benhiba, and Mohammed Janati. 2017. From Periphery to Core: A Temporal Analysis of GitHub Contributors' Collaboration Network. In *Collaboration in a Data-Rich World: Proceedings of the 18th IFIP WG 5.5 Working Conference on Virtual Enterprises (PRO-VE 2017)*. Springer International Publishing, 217–229. https://doi.org/10.1007/978-3-319-65151-4_21

[17] Raphaël Faudou, David Sciamma, Patrick Hofer, Sébastien Gérard, Etienne Juliot, Lucas Bigeardel, Kenn Hussey, Freddy Allilaire, Nick Boldt, Pierre Gaufillet, and Jacques Lescot. 2008. MDT/Papyrus-Proposal. Online. (2008). http://wiki.eclipse.org/MDT/Papyrus-Proposal Last visited: 2017-08-20.

[18] Brian Fitzgerald. 2006. The Transformation of Open Source Software. *Management Information Systems Quarterly* 30, 3 (Sept. 2006), 587–598.

[19] Brian Fitzgerald, Klaas-Jan Stol, Sten Minör, and Henrik Cosmo. 2017. *Scaling a Software Business: The Digitalization Journey.* Springer International Publishing, Cham, Switzerland. https://doi.org/10.1007/978-3-319-53116-8

[20] Jonas Gamalielsson and Björn Lundell. 2014. Sustainability of Open Source Software Communities Beyond a Fork: How and Why Has the LibreOffice Project Evolved? *Journal of Systems and Software* 89 (2014), 128–145. https://doi.org/10.1016/j.jss.2013.11.1077

[21] Jonas Gamalielsson, Björn Lundell, and Anders Mattsson. 2011. Open Source Software for Model Driven Development: A Case Study. In *Open Source Systems: Grounding Research - Proceedings of the 7th IFIP WG 2.13 International Conference.* Springer, 348–367. https://doi.org/10.1007/978-3-642-24418-6_30

[22] Matt Germonprez, Julie E. Kendall, Kenneth E. Kendall, and Brett Young. 2014. Collectivism, Creativity, Competition, and Control in Open Source Software Development: Reflections on the Emergent Governance of the SPDX® Working Group. *International Journal of Information System Management* 1, 1/2 (June 2014), 125–145. https://doi.org/10.1504/IJISAM.2014.062290

[23] Simon Grand, Georg von Krogh, Dorothy Leonard, and Walter Swap. 2004. Resource Allocation Beyond Firm Boundaries. *Long Range Planning* 37, 6 (2004), 591–610. https://doi.org/10.1016/j.lrp.2004.09.006

[24] Øyvind Hauge and Sven Ziemer. 2009. Providing Commercial Open Source Software: Lessons Learned. In *Open Source Ecosystems: Diverse Communities Interacting: Proceedings of the 5th IFIP WG 2.13 International Conference on Open Source Systems, (OSS 2009).* Springer, 70–82. https://doi.org/10.1007/978-3-642-02032-2_8

[25] Björn Lundell, Jonas Gamalielsson, Stefan Tengblad, Bahram Hooshyar Yousefi, Thomas Fischer, Gert Johansson, Bengt Rodung, Anders Mattsson, Johan Oppmark, Tomas Gustavsson, Jonas Feist, Stefan Landemoo, and Erik Lönroth. 2017. Addressing Lock-in, Interoperability, and Long-Term Maintenance Challenges Through Open Source: How Can Companies Strategically Use Open Source?. In *Open Source Systems: Towards Robust Practices - Proceedings of the 13th IFIP WG 2.13 International Conference, (OSS 2017).* Springer, 80–88. https://doi.org/10.1007/978-3-319-57735-7_9

[26] Björn Lundell, Brian Lings, and Edvin Lindqvist. 2010. Open Source in Swedish Companies: Where are we? *Information Systems Journal* 20, 6 (2010), 519–535. https://doi.org/10.1111/j.1365-2575.2010.00348.x

[27] Björn Lundell and Frank van der Linden. 2013. *Open Source Software as Open Innovation: Experiences from the Medical Domain.* Springer, Berlin/Heidelberg, 3–16. https://doi.org/10.1007/978-3-642-31650-0_1

[28] MariaDB Foundation. 2017. Getting Started for Developers. Online. (2017). https://mariadb.org/get-involved/getting-started-for-developers/ Last visited: 2017-08-11.

[29] MariaDB Foundation. 2017. Governance. Online. (2017). https://mariadb.org/about/governance/ Last visited: 2017-09-22.

[30] MariaDB Foundation. 2017. MariaDB Contributor Agreement. Online. (2017). https://mariadb.org/get-involved/getting-started-for-developers/mca/ Last visited: 2017-08-15.

[31] M. Lynne Markus. 2007. The Governance of Free/Open Source Software Projects: Monolithic, Multidimensional, or Configurational? *Journal of Management & Governance* 11, 2 (May 2007), 151–163. https://doi.org/10.1007/s10997-007-9021-x

[32] Mike Milinkovich. 2017. Open Collaboration, the Eclipse Way. Online. (2017). https://www.slideshare.net/mmilinkov/icse-2017-keynote-open-collaboration-at-eclipse Last visited: 2017-09-22.

[33] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. 2002. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering Methodology* 11, 3 (July 2002), 309–346. https://doi.org/10.1145/567793.567795

[34] Papyrus Industry Consortium. 2017. Papyrus IC. Online. (2017). https://wiki.polarsys.org/Papyrus_IC Last visited: 2017-08-11.

[35] PolarSys. 2017. Open Source Solutions for Systems Engineering and Embedded Systems. (2017). https://www.polarsys.org/ Last visited: 2017-08-11.

[36] Sylvie Robert. 2007. New Trends and Needs for Avionics Systems. In *ARTEMIS Conference.*

[37] Maha Shaikh and Ola Henfridsson. 2017. Governing Open Source Software Through Coordination Processes. *Information and Organization* 27, 2 (2017), 116–135. https://doi.org/10.1016/j.infoandorg.2017.04.001

[38] Vandana Singh. 2012. Newcomer Integration and Learning in Technical Support Communities for Open Source Software. In *Proceedings of the 17th ACM International Conference on Supporting Group Work (GROUP '12).* ACM, New York, NY, USA, 65–74. https://doi.org/10.1145/2389176.2389186

[39] Megan Squire. 2017. Considering the Use of Walled Gardens for FLOSS Project Communication. In *Open Source Systems: Towards Robust Practices - Proceedings of the 13th IFIP WG 2.13 International Conference, (OSS 2017).* 3–13. https://doi.org/10.1007/978-3-319-57735-7_1

[40] Michael Widenius. 2017. How to Create a Successful (in Business and Development) Open Source Project. Online. (Sept. 2017). http://www.ipexponordic.com/Speakers-2017/Monty-Widenius Last visited: 2017-09-22.

[41] Minghui Zhou and Audris Mockus. 2015. Who Will Stay in the FLOSS Community? Modeling Participant's Initial Behavior. *IEEE Transactions on Software Engineering* 41, 1 (Jan 2015), 82–99. https://doi.org/10.1109/TSE.2014.2349496

[42] Minghui Zhou, Audris Mockus, Xiujuan Ma, Lu Zhang, and Hong Mei. 2016. Inflow and Retention in OSS Communities with Commercial Involvement: A Case Study of Three Hybrid Projects. *ACM Transactions on Software Engineering Methodology* 25, 2, Article 13 (April 2016), 13:1–13:29 pages. https://doi.org/10.1145/2876443

[43] Pär J. Ågerfalk and Brian Fitzgerald. 2008. Outsourcing to an Unknown Workforce: Exploring Opensourcing as a Global Sourcing Strategy. *Management Information Systems Quarterly* 32, 2 (2008), 385–409.

A DATA SOURCES

A.1 Apache Solr

- Website: http://lucene.apache.org/solr/
- Wiki: https://wiki.apache.org/solr/FrontPage
- Developer mailing list: http://mail-archives.apache.org/mod_mbox/lucene-dev/
- Solr user mailing list: http://mail-archives.apache.org/mod_mbox/lucene-solr-user/
- Jira: https://issues.apache.org/jira/projects/SOLR

A.2 Bouncy Castle

- Website: https://www.bouncycastle.org/
- Wiki: https://www.bouncycastle.org/wiki/
- Mailing list: https://www.bouncycastle.org/devmailarchive/index.html
- GitHub C#: https://github.com/bcgit/bc-csharp
- GitHub Java: https://github.com/bcgit/bc-java
- Jira: http://www.bouncycastle.org/jira/secure/Dashboard.jspa

A.3 Eclipse Leshan

- Website: http://www.eclipse.org/leshan/
- Wiki: https://github.com/eclipse/leshan/wiki
- Mailing list: http://dev.eclipse.org/mhonarc/lists/leshan-dev
- GitHub: https://github.com/eclipse/leshan

A.4 MariaDB

- Website: https://mariadb.org/
- Developer mailing list: https://launchpad.net/~maria-developers
- Discussion mailing list: https://launchpad.net/~maria-discuss
- GitHub: https://github.com/MariaDB/server
- Jira: https://jira.mariadb.org/projects/MDEV/issues

A.5 Papyrus

- Website: https://eclipse.org/papyrus/
- Wiki: http://wiki.eclipse.org/Papyrus
- Forum: http://www.eclipse.org/forums/index.php/f/121/
- Developer mailing list: http://dev.eclipse.org/mhonarc/lists/mdt-papyrus.dev
- Bugzilla: https://bugs.eclipse.org/bugs/describecomponents.cgi?product=Papyrus
- Gerrit: https://git.eclipse.org/r/#/q/project:papyrus/org.eclipse.papyrus
- Git Repository: http://git.eclipse.org/c/papyrus/org.eclipse.papyrus.git/