An Open Source Environment for an Agile Development Model

Paolo Ciancarini[1,2(✉)], Marcello Missiroli[1], Francesco Poggi[3], and Daniel Russo[4]

[1] DISI, University of Bologna, Bologna, Italy
paolo.ciancarini@unibo.it
[2] Innopolis University, Innopolis, Russian Federation
[3] DCE, University of Modena and Reggio Emilia, Modena, Italy
francesco.poggi@unimore.it
[4] Department of Computer Science, Aalborg University, Aalborg, Denmark
daniel.russo@cs.aau.dk

Abstract. Tools are of paramount importance in automating software engineering tasks; although the Agile Manifesto prefers "*individuals and their interactions over processes and tools*", some agile development activities make no exception and can be automated effectively and successfully. In process frameworks like Scrum or similar ones some activities are in fact quite structured and need specific tool support. Hence, it is interesting to study the combination of specific agile practices with OSS tools.

In this paper we introduce the Compositional Agile System (CAS), an environment created to support iAgile and automate some of its tasks using OSS tools. iAgile is a Scrum-like model designed to develop critical systems in the military domain.

1 Introduction

The relationship between a development method and the tools which compose its programming environment has been debated for a long time. In a seminal paper Osterweil introduced the idea that software processes should be programmed just like applications [27], since they "*are software, too*". This idea is powerful and had the consequence to foster the study of the so called "process engines", namely environments programmed in order to support specific development methods [2,15].

The Agile era inaugurated by the Agile Manifesto gave less importance to tools, in fact the first of the four Agile values says: "*(...we have come to value...) Individuals and interactions over processes and tools*".

However the development of modern, complex software needs a lot of careful effort; several specific activities can and should be automated. For instance, the management of the product backlog in Scrum is well supported by digital taskboards like Trello [26]. Even the concept of *Definition of Done* can be partially automated [16] using for instance some tool for checking the quality of code

Published by Springer Nature Switzerland AG 2020
V. Ivanov et al. (Eds.): OSS 2020, IFIP AICT 582, pp. 148–162, 2020.
https://doi.org/10.1007/978-3-030-47240-5_15

and measuring the technical debt, like SonarQube. Moreover, the globalization of the software industry has fostered the development of collaborative platforms - like Jira - in multi-site environments to reduce costs and increase productivity.

Thus, it is still interesting to study what are the most appropriate approaches and tools to support agile developments. Another, complementary, question is if the interactions necessary to any team-based software development method suggest to design specific ad-hoc platforms.

In this paper we introduce the Compositional Agile System (CAS in short), an environment created to support iAgile, an improved agile development model introduced to develop critical systems in the military domain.

The structure of this paper is the following: in the next Sect. 2 we describe the iAgile method for developing critical systems in specific domains; in Sect. 3 we discuss the requirements of CAS, an open source development environment supporting a Scrum-like model called iAgile; in Sect. 4 we describe the CAS architecture. In Sect. 5 we compare CAS with other environments and services for agile developments. Finally, in Sect. 6 we describe our future plans and draw our conclusions.

2 iAgile: An Improved Agile Development Model for the Military Domain

Eisenhower is attributed the following quote: "*In preparing for battle I have always found that plans are useless, but planning is indispensable*". This applies to Agile approaches like Scrum, where planning the overall project is forbidden, however planning sprint-by-sprint is mandatory. iAgile is a model for empirical project management of software developments. Inspired by the Agile Manifesto, it has been derived from Scrum, and has been especially tailored for Command and Control (C2) critical systems in the military domain [23]. A C2 system in this domain includes functional military areas to cope with humanitarian assistance, stability operations, counterinsurgency operations, and combat operations [25].

The transition to Agile was needed to support faster adaptation to changes to operational needs and quality and security requirements but was also mandated by a drastic reduction in the budgets experienced in Italian public administration. The continual evolution of the operational environment in conflict theaters generates instability of the C2 requirements; this means that their developers have to work at all times with unstable and unconsolidated mission needs. Thus, planning a C2 system using the old, strongly planned, waterfall approach became unsuitable [9].

The initial idea was to experiment an in-house development of some novel functions of an established, waterfall-based, C2 system. The results using Scrum with some developers rented for specific enhancements were encouraging, however the stakeholders felt that some productivity analysis was necessary. In fact, some aspects of Scrum were not satisfactory for the context of C2 systems developed and operated in-house.

The introduction of an Agile approach in the development of critical software required the solution of many problems and the construction of a solid structure based on four principles: user community governance, specific Agile training, new Agile CASE tools and custom Agile development doctrine.

Thus for the new developments a method called iAgile was introduced [23]. Table 1 shows how the Agile principles[1] have been embedded in iAgile.

Table 1. iAgile embeds agile principles

Agile principles	iAgile
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software	iAgile has been introduced in order to develop LC2EVO, a military C2 system, aiming at shortening the "time-to-the-field" of new functions [24]
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage	iAgile systematically developed new incremental versions which were immediately tested on the field [25]
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale	Sprints lasted 4 weeks + 1 for certification [6]
Business people and developers must work together daily throughout the project	Developers and users worked daily together [3]
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done	iAgile has been quite successful, exploting the enthusiasm of both civil and military developers [7]
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation	Daily meetings and physical shared taskboard were used systematically [4]
Working software is the primary measure of progress	The LC2EVO project executed about 30 sprints, each delivering a working increment [17]
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely	The project lasted 18 months, produced several releases with a string reduction of development costs [8]
Continuous attention to technical excellence and good design enhances agility	iAgile requires excellent technical capabilities [14]
Simplicity–the art of maximizing the amount of work not done–is essential	The "lean" approach implicit in iAgile shortened the workflow necessary to implement critical requirements [18]
The best architectures, requirements, and designs emerge from self-organizing teams	The quality of the final system is high and is still being used [17]
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly	Projects based on iAgile used systematically retrospectives [14]

[1] https://agilemanifesto.org/iso/en/principles.html.

3 Requirements of a Development Environment for iAgile

The project we were involved consisted in defining a development environment supporting the iAgile development model. For security reasons we could not use commercial software online in the cloud. The target of the project was to develop an open source development environment suitable to be deployed either on a private network or in a hybrid cloud keeping complete control on its implementation and extensions.

We identified the following user stories (or epics) for CAS:

1. As a stakeholder, I am interested in estimating and monitoring the effort required by a new function to be added to the system, in order to know how many people are necessary and their productivity.
2. As a Product Owner during the project I need to check productivity data in order to predict when a release will be possible, so that *delivery on time* is ensured.
3. As a developer or stakeholder I need to communicate easily, fast, and safely with all other developers or stakeholders.
4. As a PO I need to check the code delivered by the team with respect to the Product Backlog (*percentage of adopted work*), including the Definition of Done.
5. As a developer I need tool support for asymmetric pair programming in order to get help from senior developers or domain experts.
6. As a developer I need tool support to check the quality of my code in order to satisfy the Definition of Done.
7. As a developer I need tool support to monitor defects in libraries and open source components.
8. As a PO I need to track how the user stories of the Product Backlog evolved into code in order to plan the next release.

These User Stories do not cover the whole spectrum of iAgile roles and activities, however we have found them useful as an initial reference that now can be expanded.

3.1 Mapping the Requirements on Open Source Components

The above user stories were suggested by interviewing the stakeholders involved in the development of a critical system. In particular, the introduction of the COSYSMO estimation framework [28] was suggested by some military stakeholders who were interested in matching system engineering estimation with agile development of software-intensive components.

In the meantime in the same project we had proposed a survey to the agile community; the results of such survey are reported in [10]. According to the survey, Jira is currently the most popular tool for agile developments. Also Trello and Slack are very popular. However these tools are not adequate for the context of our project, since they are commercial and based on proprietary clouds.

Table 2. Mapping the user stories and main iAgile roles on some open source tools

#	User story	Tool	iAgile roles
US1	Estimating and monitoring	COSYSMO, COCOMO	AD
US2	Checking the team productivity	Taiga, GitInspector, logger plug-in	PO, SM, team, AD
US3	Supporting communication among all stakeholders	Mattermost	team, PO, AD
US4	Check the code	Taiga, Git (GitLab CE), Junit	PO, team
US5	Asymmetric remote pair programming	Saros (plugin Eclipse)	team, SM
US6	Check code quality and technical debt	SonarQube server	team, SM, (PO)
US7	Issue tracking and library monitoring	Bugzilla, SonarLint (plugin Eclipse)	team, (PO, AD)
US8	Track user stories into code and releases	Taiga	PO, team

Thus we searched the open source market and identified a number of open source components useful to satisfy the user stories listed above. The result of our study is shown in Table 2.

We decided to include these components as services in the CAS back end. We also added a front end in the form of an IDE like Eclipse [19] or IntelliJ IDEA [20].

4 Architecture of CAS

The Compositional Agile System (CAS) is based on two subsystems: the *CAS server*, whose architecture is based on microservices, and the *CAS client*, a normal IDE like Eclipse or IntelliJ IDEA (we are working also on an Atom client) enriched with some plugins.

Figure 1 shows the CAS architectural stack, available on any operating system able to host dockerized services. Each CAS server service (e.g. SonarQube, Taiga, GitLab, etc.) - represented by its own logo - has its own container, which includes a locale storage system. The environment architecture is compositional in the sense that each service is separate and new services can easily be added, following the approach to *agile software architecture* suggested in [5].

4.1 The CAS Client

By CAS client we mean any IDE that developers use as the front end of the development environment. We have integrated two IDEs with CAS: Eclipse and IntelliJ IDEA. We are also working at integrating other IDEs such as Atom, an IDE developed and integrated with GitHub.

The integration of each client consists of an original plug-in we developed for collecting developers data, in order to monitor productivity [13], plus other plugins available for the main CAS server services and necessary to support some user story.

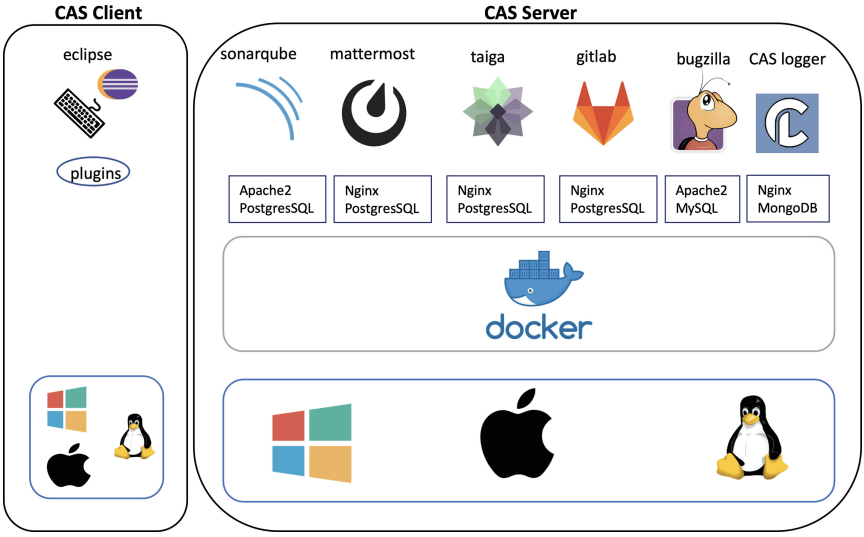

Fig. 1. CAS client-server architecture: the CAS server lower layer is the operating system, then the docker layer, and finally the services, each with its own web server and storage dbms. The CAS client components (i.e. Eclipse, which is shown in the figure, or IntelliJ IDEA) and their plugins are not dockerized. They rely on CAS server components to store and analyze developers' data.

Eclipse: Eclipse is an integrated multi-language and cross-platform development environment. It is designed by a consortium called the Eclipse Foundation, which includes large companies such as IBM, Ericsson, HP, Intel, SAP. Eclipse is a free and open source software distributed under the terms of the Eclipse Public License.

Eclipse can be used for the development of a huge variety of software products. It can be used as a complete IDE for several programming languages, for instance: Java Development Tools (JDT) for the Java language, C/C++ Development Tools (CDT) the C/C++ language. Similarly, through plugins it is possible to manage XML, JavaScript, PHP, as also designing graphically a GUI for a Java application (i.e., Window Builder). Eclipse is considered an environment for Rapid Application Development.

The platform is focused on the use of plugins, which are software components designed for a specific purpose. Indeed, the whole platform is a set of plugins, including the basic version, and anyone can develop and edit the different plugins. In the basic version it is possible to program in Java, taking advantage of help functions such as automatic completion ("Code completion"), suggestion of the types of parameters of the methods, direct access to git and automatic rewriting of the code ("Refactoring function") in case of changes in the classes.

Finally, since Eclipse has been written in Java, it is portable and available for Linux, macOS, Windows and other platforms.

IntelliJ IDEA: IntelliJ IDEA (it was once known simply as IntelliJ) is an integrated development environment for the Java programming language. It has been developed by JetBrains and is available in both Apache and commercial proprietary editions. The first version of IntelliJ IDEA was the first IDE to integrate features such as code navigation and *code refactoring*. In a 2010 ranking, IntelliJ received the highest test score among the four most popular Java programming tools: Eclipse, IntelliJ IDEA, NetBeans and JDeveloper[2].

In December 2014, Google announced version 1.0 of Android Studio, an open source IDE for developing Android apps, based on the Community edition of IntelliJ IDEA[3]. IntelliJ IDEA has two editions: Community Edition (OSS) and Ultimate Edition (which is the proprietary version).

4.2 Extending a CAS Client

One of the main reasons for choosing Eclipse/IntelliJ is the large amount of plugins existing for both environments. The main examples are *saros*, *egit* and *sonarlint*.

Another reason is that every developer can build specific plugins in order to add new functions to her IDE. For example, we have introduced a keylogger plugin to record data concerning the actions performed by the developers.

The *saros* plugin allows remote pair programming, and supports synchronous text editing on a document shared by up to five devlopers.

The *egit* plugin integrates either GitLab or GitHub directly inside Eclipse or IntelliJ.

The *sonarlint* plugin allows to analyze the quality of the source code and it reports any problems during the editing.

The *logger* plugin helps developers to record and then to monitor and summarize their actions. Accordingly, the PO can analyze the productivity of the team or individual developers.

4.3 CAS Services

GitLab: Git is an open source Version Control System (VCS), also used as Software Configuration Management (SCM) tool. Git allows to keep track of the changes made to the files of a project, keeping the history of all the changes made, so that it is always possible to access the different versions of the project's artifacts [22]. Git is more precisely a system for distributed version control (DVCS), since it allows a group of people to collaborate on the same file system, sharing it and keeping track of who does what.

It is always possible to restore, partially or completely, the structure of the base directory of a project so that there is a version of the files and subfolders with the changes that had been made at some time in the past.

[2] www.infoworld.com/article/2683534/infoworld-review--top-java-programming-tools.html.

[3] www.venturebeat.com/2014/12/08/google-releases-android-studio-1-0-the-first-stable-version-of-its-ide.

Git is especially useful when working with plain text files, such as source code. In fact, it provides various tools to compare the different versions of a file and see what changes have been made between one version and another. Although numerous graphical interfaces are available, Git is usually used via the command line interface. For this reason it mostly suggested for software development, but could be also used in other areas. For example, it can be useful if a group of people write a book collaboratively using Microsoft Word or Latex.

The CAS server includes GitLab, a Git-repository manager providing a rich set of functionalities such as tools for DevOps lifecycle support, issue-tracking, and Continuous Integration/Continuous integration Deployment (CI/CD) pipeline features.

Bugzilla: Bugzilla is an issue tracker, a program that creates a service to record and manage problem reports. Bugs can be submitted to the system by anyone, and will be assigned to the developer who has been associated with the bug. Bugs can have various types of status associated with them, along with user comments and examples of the error itself.

Mattermost: Mattermost is a scalable messaging system for team communication. It allows both direct and group-based communication. It supports topic or meeting-based channels. It offers string search functionality in messages and channels. Mattermost facilitates communication between teams by keeping all messages in one place. Those can be easily searched within the repository and are available for every team member everywhere. It is also possible to create an own chat solution hosted on a private cloud service.

In CAS, Mattermost is the main communication system, useful for the various iAgile ceremonies, especially if they are performed remotely and e.g., the PO is not co-located with the team such as in some sprint planning, daily meeting, sprint review or retrospective events.

SonarQube: An important issue in iAgile - just like in Scrum - is the concept of Definition of Done. We have introduced in iAgile the concept of *Definition of Done* based on a tool for checking the quality of code. The tool we have chosen is SonarQube.

Traditional approaches to software quality management tend to postpone code testing until after its completion, and to complete it at the end of the development of the entire software. This kind of approach, at best, generates delays and the need to review the whole work, when even the smallest bug is found. In the worst case scenario, on the other hand, it pushes the delivery of low quality software with several defects, and difficult to maintain. To address this issue, Continuous Inspection is the goal of SonarSource, a company that focuses on software quality analysis, which goal is to enhance software quality along the entire development process. Accordingly, the tool SonarQube which is a service for the ongoing management of the technical debt has been developed to support Continuous Inspection.

SonarQube is an open-source program for continuous analysis of code quality. It is able to perform automatic static analysis of code to detect bugs, code smells, and security vulnerabilities on several programming languages. It offers reports on duplicated code, coding standards, unit tests, code coverage, code complexity, comments, bugs, and security vulnerabilities. It also integrates with Eclipse and IntelliJ IDEA through the SonarLint plug-in.

A way to enrich a Definition of Done is via coding rules, namely rules that developers have to follow when writing their code. There are two ways to extend coding rules in SonarQube. The first is by writing coding rules using Java via a SonarQube plugin; the second is by adding XPath rules directly through the SonarQube web interface. The Java API is richer than the XPath one.

SonarQube might therefore be considered as a reference tool for analyzing the quality of software both continuously during development and postmortem at the end of the project.

The tool can be used both as a cloud service (i.e., SonarCloud) and as an open source package that can be hosted on own proprietary servers. In CAS we used the open source version, that has been included as a component of the CAS server. One of the most interesting advantages of SonarQube is the ability to track the evolution of a project's quality metrics over time. Every time an inspection is performed, users have a visual representation of the project's quality. It provides a snapshot of the code quality, including lagging indicators (problems already present) and leading indicators (problems that may arise in the future). Furthermore, it also provides services to address future issues, such as the code-review tools.

Taiga: Taiga is a free and open-source project management system able to host a product backlog and the related items of agile processes, like sprint backlogs and burndown charts. Backlogs are shown as a running list of all User Stories and related tasks belonging to the project. It is released under GNU Affero General Public License.

Taiga tracks the progress of a project using a taskboard whose template can be either Kanban or Scrum, or both. For iagile we use the Scrum template.

Taiga integrates chats and video conferencing functions with the use of third party services. The on-premise version of Taiga can be downloaded from its github repositories and used for free. This project management system can interface with web-based version control repositories like GitHub and Bitbucket. Taiga also provides several importers to facilitate migration from other proprietary software platforms, like Trello.

The project template can be configured using the following modules: Epics, Backlog, Kanban, Issues, Wiki, Meet Up. For Scrum active are: Backlog, Issues and Wiki; for Kanban the Kanban Board suffices. Combining the Backlog module and the Kanban Board we get a valid template for Scrumban.

The roles supported are: Product Owner, UX, Design, Front, Back, Stakeholder, External User. All roles except External User must be registered users.

Each role gets specific access privileges: Epics, Sprints, User story, Tasks, Issues, Wiki. Access rights are read-only, new-item, modify-item, comment-item, delete-item.

4.4 CAS Administration

Currently, the CAS administration consists mainly in controlling accesses to CAS services. Each service in CAS has its own access protocol. We plan to overcome this weakness in the next version.

Figure 2 shows the configuring process for CAS. First the server settings are defined, then a CAS View is added to the IDE Eclipse, then a new CAS instance can be used.

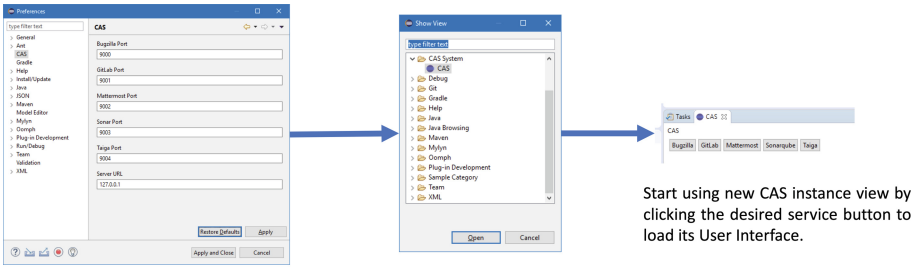

Fig. 2. Configuring CAS

5 Comparison

We can compare CAS to other tools chosen by agile teams for software developments. The main contender is Jira, that is arguably the most comprehensive and popular tool used in software project management.

Jira: Jira is a comprehensive issue tracking and project management system. It is, however, neither free nor open source.

It can be accessed either as a cloud service or installed locally. Jira provides a variety of services and, on top of it, one can add a series of tightly integrated tools to further expand. To have a meaningful comparison, we chose the Jira+Helpdesk offer, the one that maps CAS services more precisely.

Given the tight integration and the single sign-on features, installation and setup is far easier and straightforward with respect to CAS, which currently requires separate access configurations for the various services.

Taiga vs Jira: Like Taiga, Jira also offers both Kanban and Scrum taskboard support. It is possible to setup User Stories, estimate them, assign them to specific persons. Thereafter, it is possible to start a Sprint or the Kanban board. It also offers a "Roadmap" page that presents a higher level of abstraction (Epics, for example). Several reporting options are avaialable. However, Taiga got inspiration from Trello. Both value simplicity and creativity and require low learning effort from the users. Moreover Taiga supports interactions via its telephone app for iOS, Android, and Windows devices.

GitLab vs Bitbucket: Bitbucket is one of the main Git-based service for cooperative source management recommended for Jira; its features match closely those of GitLab and other similar services. It can be integrated in Jira using a single sign-on mechanism, though the integration is not very tight.

Bugzilla vs Jira: Jira's name is a corruption of bugzilla, which was used by the Jira team before they produced Jira. In fact, originally Jira was a ticking service, and therefore this section is very mature and well integrated with the project management section compared to bugzilla.

Sonarqube and Mattermost vs Jira: These services have no direct equivalent in the Atlassian platform. Sonarqube can be configured to work with Jira, using procedures similar to CAS, but no tight integration is possible. After discontinuing its internal instant communication services (Hipchat) Atlassian suggests to use Slack, arguably the most famous and mature service of its kind which is however not open source or free.

6 Conclusions and Future Works

The CAS environment has been developed in four months, and is currently used in research projects [11,12] and by groups of students. So it has been tested for a very short time: the preliminary evaluations by the students are encouraging.

An important question that we should answer closing this paper is: which is the difference between iAgile and Scrum, and which impact has on development tools? This is a relevant question that has been already partially answered in [6,7]. We now offer a new discussion after having built CAS.

Table 3. Comparing Scrum vs iAgile/CAS

Scrum	iAgile
Roles: PO, SM, team	New roles: Requirement Owner, Operative PO, PO team, expert in cybersecurity
ScrumMaster	Military Facilitator
Self organizing team	Team with both civil and military personnel, including at least one cybersecurity expert
Contract: free	Contract: in-house or outsourced
Simple user stories	Structured user stories
Most popular tool: Jira (commercial)	Main tool: CAS (open source)
Shared physical taskboard	Digital taskboard (Taiga)
Face to face communication	Digital channels (Mattermost)
Metrics: chosen by the team	COSYSMO + development metrics
Definition of Done: defined by the PO	Definition of Done including SonarQube analysis
Various scalability models	LeSS-based scrum of scrums
Product certification: difficult	Product certification in each sprint

Table 3 shows a comparison between Scrum and iAgile. After we built CAS we have appreciated how strongly connected are a development model and its environment. For instance, we have yet to solve the problem of supporting scrums of scrums according to the LeSS model [21] using Taiga.

Another important issue is the combination of OSS licenses. The tools we have used have the licenses listed in Table 4.

Table 4. OSS licenses used in CAS

Component	License	Version	Link
Eclipse	EPL	4.10	www.eclipse.org/eclipse/news/4.10/
Taiga	GNU AGPL v3	4.0.0	github.com/taigaio/
GitLab	MIT	11.11	about.gitlab.com/releases/
Egit	BSD	5.3	www.eclipse.org/egit/
Gitinspector	GPL v3	0.4.4	github.com/ejwa/gitinspector/releases
Mattermost	Apache 2.0	5.11	docs.mattermost.com/administration/changelog
Saros	GNU v2	15.0	www.saros-project.org/releases/
SonarQube	LGPL v3	7.7	www.sonarqube.org/downloads/
bugzilla	GPL (MPL)	5.0.6	www.bugzilla.org/releases/
PostGres	PostgreSQL	9.5/9.6	dev.mysql.com/doc/relnotes/mysql/8.0/
docker	Apache 2.0	18.09.6	docs.docker.com/engine/release-notes/

We note that all these licenses are different, so we have to study their composition [1].

Another issue that we have to study is how the evolution of the different services proceeds smoothly under Docker and with respect the other services included in the CAS. For instance, SonarQube evolves rapidly and its API changes.

Acknowledgements. We thank prof. gen. Angelo Messina (rit.) for the interviews and the lively constructive discussions we had concerning iAgile. We thank for the support obtained with the project PNRM AMINSEP from CINI and from CNR/ISTC.

References

1. Alspaugh, T.A., Asuncion, H.U., Scacchi, W.: Analyzing software licenses in open architecture software systems. In: ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development, pp. 54–57. IEEE (2009)
2. Ambriola, V., Ciancarini, P., Corradini, A.: Declarative specification of the architecture of a software development environment. Softw. Pract. Exp. **25**(2), 143–174 (1995)
3. Arseni, G.: Role of the design authority in large scrum of scrum multi-team-based programs. In: Ciancarini, P., Sillitti, A., Succi, G., Messina, A. (eds.) Proceedings of 4th International Conference in Software Engineering for Defence Applications. AISC, vol. 422, pp. 181–189. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-27896-4_15
4. Aslam, H., Brown, J.A., Messina, A.: Affordance theory applied to agile development: a case study of LC2EVO. In: Ciancarini, P., Mazzara, M., Messina, A., Sillitti, A., Succi, G. (eds.) SEDA 2018. AISC, vol. 925, pp. 24–35. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-14687-0_3
5. Babar, M., Brown, A., Mistrik, I. (eds.): Agile Software Architecture. Morgan Kaufmann, Boston (2014)
6. Benedicenti, L., Messina, A., Sillitti, A.: iAgile: mission critical military development. In: Proceedings of IEEE International Conference on High Performance Computing and Simulation, pp. 545–552. Genoa, Italy (2017)
7. Benedicenti, L., Ciancarini, P., Cotugno, F., Messina, A., Sillitti, A., Succi, G.: Improved agile: a customized scrum process for project management in defense and security. In: Mahmood, Z. (ed.) Software Project Management for Distributed Computing. CCN, pp. 289–314. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-54325-3_12
8. Chang, S.J., Messina, A., Modigliani, P.: How agile development can transform defense IT acquisition. In: Ciancarini, P., Sillitti, A., Succi, G., Messina, A. (eds.) Proceedings of 4th International Conference in Software Engineering for Defence Applications. AISC, vol. 422, pp. 13–26. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-27896-4_2
9. Ciancarini, P., Messina, A., Poggi, F., Russo, D.: Agile knowledge engineering for mission critical software requirements. In: Nalepa, G.J., Baumeister, J. (eds.) Synergies Between Knowledge Engineering and Software Engineering. AISC, vol. 626, pp. 151–171. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-64161-4_8

10. Ciancarini, P., Missiroli, M., Sillitti, A.: Preferred tools for agile development: a sociocultural perspective. In: Mazzara, M., Bruel, J.-M., Meyer, B., Petrenko, A. (eds.) TOOLS 2019. LNCS, vol. 11771, pp. 43–58. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29852-4_3
11. Ciancarini, P., Poggi, F., Rossi, D., Sillitti, A.: Improving bug predictions in multicore cyber-physical systems. In: Ciancarini, P., Sillitti, A., Succi, G., Messina, A. (eds.) Proceedings of 4th International Conference in Software Engineering for Defence Applications. AISC, vol. 422, pp. 287–295. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-27896-4_24
12. Ciancarini, P., Poggi, F., Rossi, D., Sillitti, A.: Analyzing and predicting concurrency bugs in open source systems. In: Proceedings of International Joint Conference on Neural Networks (IJCNN), pp. 721–728. IEEE (2017)
13. Coman, I.D., Sillitti, A., Succi, G.: A case-study on using an automated in-process software engineering measurement and analysis system in an industrial environment. In: Proceedings of 31st International Conference on Software Engineering, pp. 89–99. IEEE (2009)
14. Cotugno, F.R.: Managing increasing user needs complexity within the ITA army agile framework. In: Ciancarini, P., Sillitti, A., Succi, G., Messina, A. (eds.) Proceedings of 4th International Conference in Software Engineering for Defence Applications. AISC, vol. 422, pp. 1–11. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-27896-4_1
15. Cugola, G., Ghezzi, C.: Software processes: a retrospective and a path to the future. Softw. Process Improv. Pract. **4**(3), 101–123 (1998)
16. Diebold, P., Ostberg, J.-P., Wagner, S., Zendler, U.: What do practitioners vary in using scrum? In: Lassenius, C., Dingsøyr, T., Paasivaara, M. (eds.) XP 2015. LNBIP, vol. 212, pp. 40–51. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18612-2_4
17. Galantini, L., Messina, A., Ruggiero, M.: Software requirements complexity analysis to support the "advisory network in to the nation forces build-up". In: Ciancarini, P., Mazzara, M., Messina, A., Sillitti, A., Succi, G. (eds.) SEDA 2018. AISC, vol. 925, pp. 187–197. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-14687-0_17
18. Gazzerro, S., Marsura, R., Messina, A., Rizzo, S.: Capturing user needs for agile software development. In: Ciancarini, P., Sillitti, A., Succi, G., Messina, A. (eds.) Proceedings of 4th International Conference in Software Engineering for Defence Applications. AISC, vol. 422, pp. 307–319. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-27896-4_26
19. Holzner, S.: Eclipse Cookbook. O'Reilly, Sebastopol (2004)
20. Krochmalski, J.: IntelliJ IDEA Essentials. Packt Pub (2014)
21. Larman, C., Vodde, B.: Large-Scale Scrum: More with LeSS. Addison-Wesley, Boston (2016)
22. Magana, A., Muli, J.: Version Control with Git and GitHub. Packt (2018)
23. Cotugno, F.R., Messina, A.: Adapting SCRUM to the Italian army: methods and (open) tools. In: Corral, L., Sillitti, A., Succi, G., Vlasenko, J., Wasserman, A.I. (eds.) OSS 2014. IAICT, vol. 427, pp. 61–69. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55128-4_7
24. Messina, A., Fiore, F.: The Italian Army C2 evolution: From the current SIACCON2 land command & control system to the LC2EVO using agile software development methodology. In: Proceedings of International Conference on Military Communications and Information Systems (ICMCIS), pp. 1–8. Brussels, Belgium (2016)

25. Messina, A., Fiore, F., Ruggiero, M., Ciancarini, P., Russo, D.: A new agile paradigm for mission critical software development. Crosstalk J. Def. Softw. Eng. **29**(6), 25–30 (2016)
26. Naik, N., Jenkins, P., Newell, D.: Learning agile scrum methodology using the groupware tool trello through collaborative working. In: Barolli, L., Hussain, F.K., Ikeda, M. (eds.) CISIS 2019. AISC, vol. 993, pp. 343–355. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-22354-0_31
27. Osterweil, L.: Software processes are software too. In: Proceedings of 9th IEEE International Conference on Software Engineering, pp. 2–13 (1987)
28. Valerdi, R.: The Constructive Systems Engineering Cost Model (COSYSMO). Ph.D. thesis, University of Southern California (2005)