WP3 – Architecture, Specification and Integration

D3.4.1: Component Integration, Build Management and Testing

Deliverable Lead: ASC

Contributing Partners: ASC

Delivery Date: 2014-11

Dissemination Level: Public

Final

This document describes the integration aspects of the SAM project. More precisely it describes the systems and infrastructure setup for source-code management, automatic continuous integration platform, issue tracking and fixing. Additionally it provides insight of the software development process, the integration of the different technical components and the testing process.

Document Status	
Deliverable Lead	ASC
Internal Reviewer 1	Fran Rodriguez, v0.2, TIE
Internal Reviewer 2	David Tomás, v0.3, UA
Type	Deliverable
Work Package	WP3 – Architecture, Specification and Integration
ID	D3.4.1: Component Integration, Build Management and Testing
Due Date	10.2014
Delivery Date	11.2014
Status	For Approval

Document History	
Versions	V0: ASC First draft produced by Editor V0.1: ASC Provided content to all sections V0.2: ASC 1st review version V0.3: ASC, TIE 1st review amendment, 2nd review version V0.4: ASC, UA 2nd review amendment V1.0: ASC Final version V1.01: TIE Minor edits before submission V1.02: ASC Minor edits before submission
Contributions	ASC: Norman Wessel – Document creation and all sections TIE: Fran Rodriguez, Stuart Campbell – Review of document UA: David Tomás – Review of document

Disclaimer

Project Partners

TIE Nederland B.V., The Netherlands

Ascora GmbH, Germany

Talkamatic AB, Sweden

TP Vision Belgium NV, Belgium

Institute of Communication and Computer Systems, National Technical University of Athens, Greece

The University of Reading, UK

Universidad de Alicante, Spain

Deutsche Welle, Germany

Bibliographic Data Services Limited, UK

Executive Summary

The purpose of this document is to provide the project partners with necessary information for a productive and efficient usage of the infrastructure provided in the SAM project. This includes the selection, the current status and additional information regarding the different systems (source code management, continuous integration and issue tracking). Also information about the software development process, the integration process and the testing process will be provided.

This document is divided in sections where each section contains information regarding a different infrastructure topic: Source code management, continuous integration and issue tracking are the systems which have to be discussed and selected. Also additional information will be provided so that project partners are able to use the selected systems in the right way. Additionally the chosen software development methodology will be discussed and explained. The integration results of the technical components and testing sections will be provided in the M37 deliverable, where a second version of this deliverable will be published.

As shown in Figure 1, this deliverable is closely related to the Description of Work (DOW) and the deliverable D3.3.1 – Technical Specification.

The DOW provides general information about the systems which needs to be set up to provide an infrastructure supporting the implementation phase. The deliverable D3.3.1 Technical Specification contains information and technologic decisions which affects the provision of the infrastructure and have to be considered too. Therefore this deliverable serves as a technical aide and a secondary deliverable for the work packages 4, 5, 6 and 7 where the infrastructure is of importance.

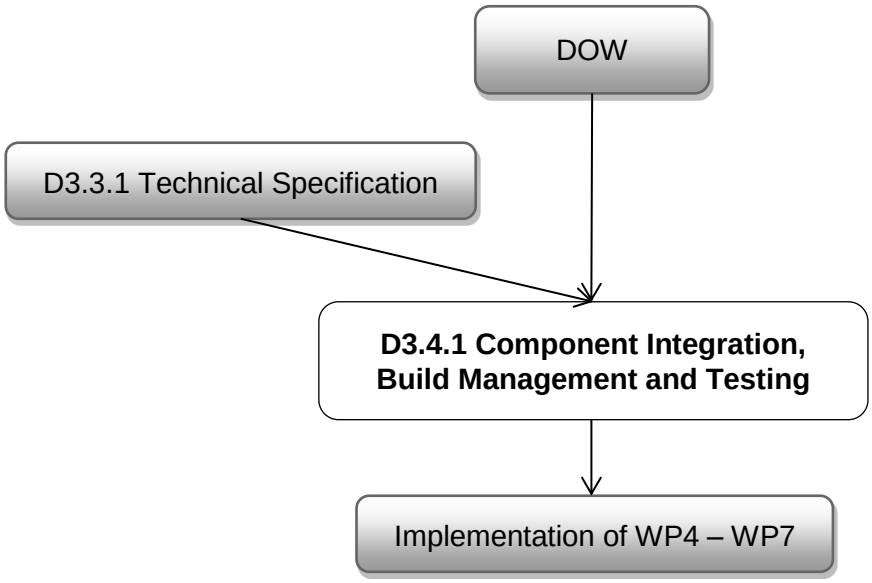

Figure 1: Context of the Deliverable

Table of Contents

1 Introduction

SAM – Dynamic Social and Media Content Syndication for 2nd Screen – is a project funded by the Seventh Framework Programme of the European Commission under Grant Agreement No. 611312. It provides a content delivery platform for syndicated data to be consumed in a contextualised social way through 2nd Screen devices.

1.1 SAM Project Overview

Today's generation of Internet-connected devices has changed the way users are interacting with media, exchanging their role from passive and unidirectional to proactive and interactive. Under this new role, users are able to comment or rate a TV show and search for related information regarding characters, facts or personalities. They do this both with friends and wider social communities through the so-called "2nd Screen".

Another coupled phenomenon is "Content Syndication" which is a field of marketing where digital content is created once and delivered too many different marketing channels (devices, social media channels, websites and stakeholders) together and so allowing efficient content control, delivery, and feedback.

However, the 2nd Screen phenomenon has grown in an unordered way. Tools are provided by the media providers companies (e.g. as mobile or tablet apps) which limits outreach and as a result, users are not stimulated and fed with relevant contextual syndicated information. European enterprises wishing to provide services have limited potential to receive feedback, which restricts the business intelligence that can be extracted and applied therefore to profit from and enrich this market.

SAM will change this disorder by developing an advanced Social Media delivery platform based on 2nd Screen and Content Syndication within a Social Media context. This is achieved by providing open and standardised ways of characterizing, discovering and syndicating media assets interactively. Users will be able to consume and prosume digital assets from different syndicated sources and different synchronised devices (e.g. connected TVs), thus creating richer experiences around the original media assets.

SAM's innovation is that instead of users reaching for the data; it is the data, which reaches the user through the syndication approach and their 2nd Screen. This is based on the creation of dynamic social communities related to the user and digital asset context (e.g. profiles, preferences and devices connected). These are dynamic hangouts where people share interests, socialise and build virtual communities. SAM will enable syndication of comments, ratings, facts, recommendations and new information that will enrich and energise the community as well as enhance personalised knowledge and satisfaction.

1.2 Deliverable Purpose, Scope and Context

The purpose of this document "Component Integration, Build Management and Testing" is to provide the project partners with necessary information for a productive and efficient usage of the infrastructure provided. This includes the selection, the current status and additional information regarding the different systems: source code management, continuous integration and issue tracking.

Source code management is the way to store, share and work with the source code generated in the implementation phase of the project. The most important feature is the

ability to work on one code base with multiple persons. Changes can be merged (does not work for binary files) so that two or more persons can work at one file at the same time without overwriting the changes of the other person.

Continuous integration (CI) is the process of combining different components to one final application, in this case combining the SAM components to the SAM platform. The goal of this process lies in the enhancement of the software quality. Changes will trigger the build of the corresponding component automatically and by using integrated software test this CI platform can reduce errors too.

Issue tracking is the process of recording, tracking and finally solving issues concerning an entity, which can be a SAM component in the SAM context. An issue can be a small bug or a new feature which should be implemented. Nonetheless in both cases the issue tracking process enables to work on these emerging issues in a more organizationally way, providing statistics and a better overview.

This deliverable also will provide information about the software development process (see Section 5), the integration process (see Section 6) and the testing process (see Section 7).

1.3 Document Status and Target Audience

This document is listed in the DOW as “public”. Nevertheless, the intended audience for this document are the technical partners within the SAM project that need to adhere to the software development practices and tools used within the project. When the tools and methodologies described in this deliverable are employed by the partners that develop software in SAM, a smooth integration of the source code and a better collaboration between partners can be ensured. The ‘real’ deliverable of course is the mentioned practice and tools which are described briefly by this document.

1.4 Abbreviations and Glossary

A definition of common terms and roles related to the realization of SAM as well as a list of abbreviations is available in the SAM Glossary.

Further information can be found at
http://wiki.socialisingaroundmedia.com/index.php/Glossary

1.5 Document Structure

This deliverable is broken down into the following sections:

- **Section 1 (Introduction):** An introduction to this deliverable including a general overview of the project, and outlines the purpose, scope, context, status, and target audience
- **Section 2 (Source Code Management):** The presentation and selection of the source code management system, the result and the current status
- **Section 3 (Continuous Integration):** The presentation and selection of the continuous integration platform, the result and the current status
- **Section 4 (Issue Tracking):** The presentation and selection of the issue tracking system, the result and the current status
- **Section 5 (Software Development):** This section provides insight into the chosen software development methodology

- **Section 6 (Integration of Technical Components):** Description on how the different SAM components will be composed to create the SAM platform
- **Section 7 (Testing):** Description and guidelines for testing procedures

- **Annexes**
  - **Annex A (Additional Information):** This section provides helpful information like tutorials and guidelines for Section 2, Section 3 and Section 4
  - **Annex B:** Table providing the URL to the different infrastructure systems selected in Section 2, Section 3 and Section 4

2 Source Code Management

Source code management (SCM) is the way to store, share and work with the source code generated in the implementation phase of the project. The secure storage of the source code is important but also how the source code can be accessed and how changes are tracked is of importance too. Tracking the changes can be done by a common feature of source code management systems.

For each provided source code entity (file) a version history is created so that changes and responsible contributors can be tracked. Also the access to these entities will be secured by different safety measures like SSH[1]. But one of the most important features is the ability to work on one code base by multiple people. Changes can be merged (does not work for binary files) so that two or more persons can work at one file at the same time without overwriting the changes of the other person. Due to the fact that multiple project partners work together, the use of such a system is required in the SAM project.

It is usually recommended to agree on one specific SCM system so partners do not have to be familiar with and maintain code in multiple systems and will have a central point where all code will be stored. Additionally the administration effort and the complexity level will be lower which also helps to be flexible when unexpected changes have to be made.

In this section the following topics will be discussed and presented:

- Presentation of eligible candidates for the SCM system
- Selection of the SCM system
- Presentation of the current status of the SCM system
- Additional information for the project partners about the procedure

The eligible candidates will be compared using different parameters which support the selection process. The following parameters have been chosen:

Parameters	Description
Maturity & Stability	Selecting a stable and mature solution may be one of the key criteria for the selection of a SCM system.
Regularly Updated	The ICT market is developing significantly faster than any other market in the world and technologies that have not been continuously updated may be considered dead. Based on this criteria,, the consortium considers it is important to choose as a base technologies that are updated regularly to reflect new market constraints.
Community	For clarifying questions and discussing possible problems of a technology, a strong community should be available.
Offline Usage	Describes if a local repository is existent and commits can be done without the need to have a connection to the remote repository.
Versioning	Versioning enables developers to keep track of changes regarding existing entities in the SCM system. This is of importance when working as a team.
GUI/Terminal Support	A developer's workspace can be very heterogeneous (Linux, Windows, MacOS) so it is important that the candidate provides tools which enables access to the repositories from different systems.
Web Frontend	When working on multiple projects from different locations, a web frontend enables a better overview and coordination of the development process.
IDE Integration	Using the same tool to implement and to provide code changes can ease the deployment process.

Figure 2: Source Code Management – Parameter Description

[1] http://en.wikipedia.org/wiki/Secure_Shell

2.1 Candidates

The following SCM systems have been chosen as eligible candidates:

- Git[2] in conjunction with GitLab[3]
- Subversion[4]
- Mercurial[5]

The chosen SCM systems are popular, but for different reasons. All of them support versioning files by tracking changes between versions, can be used terminal-based or with graphical user interfaces (GUI) and for each chosen SCM extensions for common integrated development environments (IDE) exist. All of them are stable, constantly updated and relatively bug-free. Figure 3 shows the comparison using parameters important for the later to be used platform.

Parameters	Git/GitLab	Subversion	Mercurial
Maturity & Stability	+++	++	+++
Regularly Updated	+++	+++	+++
Community	+++	++	++
Offline Usage	YES	NO	YES
Versioning	YES	YES	YES
GUI/Terminal Support	YES (both)	YES (both)	YES (both)
Web Frontend	YES	Through 3rd parties	Through 3rd parties
IDE Integration	YES	YES	YES

Figure 3: Source Code Management – Parameter Overview

Subversion (SVN) is the oldest of the three, with the highest share of developers in the consortium already familiar with the tool. SVN nevertheless has different shortcomings when compared to the other two systems. It is best described as a slightly versioned, remote file system. A central repository is used for synchronization. This means SVN can only be used when Internet connectivity is available, and it means that there is only one central place everyone is working on concurrently, which can result in more conflicts when code is merged. Past experiences had shown that branching, merging or restoring file history using SVN resulted in problems, which made its usage problematic. Even if these options exist, they are much more error prone than in the other systems. Also, SVN is much slower than the other two contenders, and wastes much more storage space. For many of these reasons, SVN will not be chosen, even if many developers are familiar with it.

The other two SCMs work in a decentralized manner. They sport a local repository that works independently of a central repository, so that commits can also be used offline to separate units of work. Branching, restoring history, merging and many other useful traits of SCM systems nevertheless work issue-free with Git and Mercurial. Much less storage space will be wasted, due to a more optimized way of handling. Due to this, Mercurial (referred to as “HG”) and Git have effectively superseded SVN in modern work environments, even though they are a bit more complex to understand.

The differences between HG and Git are very superficial, as both use the same concepts. HG is more used in Windows-development surroundings, as it is the base for “Bitbucket”, a public repository host more frequented in the Windows-related development, while Git is

[2] http://git-scm.com/
[3] https://about.gitlab.com/
[4] https://subversion.apache.org/
[5] http://mercurial.selenic.com/

much more widespread and is the basis for the public repository host GitHub. With the main IDE for Windows development, Visual Studio 2013, integrating Git over HG in the newest version, the time of HG seems well over. This is also seconded by most companies choosing Git as source code repository over other SCMs, and by more developers among the consortium being familiar with Git than with HG.

2.2 Selection

Based on the data provided in Section 2.1 the SCM system Git was selected. Git is currently state-of-the-art in this area and combined with GitLab, which helps to visualise the different projects/components, it will support the development process of the SAM components. Also Git can easily be integrated in all the partners IDEs or be used on the command line by all partners not using IDEs, provides all the necessary features expected by the SAM technical partners, and also integrates well with JIRA.

2.3 Current Status

The source code management system Git and its graphical user interface GitLab have been set up by partner ASC. GitLab is available under **http://sam.ascora.de/.** As one can see in Figure 4 only logged-in users can access the platform. All developers (RTD) have access to the system. Figure 5 shows the Dashboard of GitLab. It presents an activity stream and an overview with all existing projects.

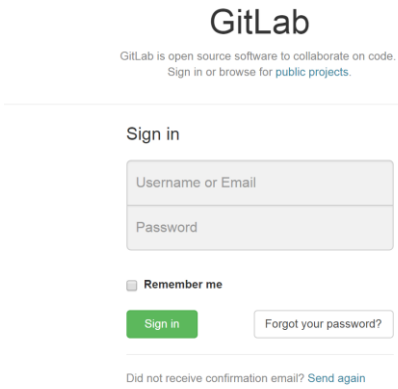

Figure 4: Source Code Management – Login Mask

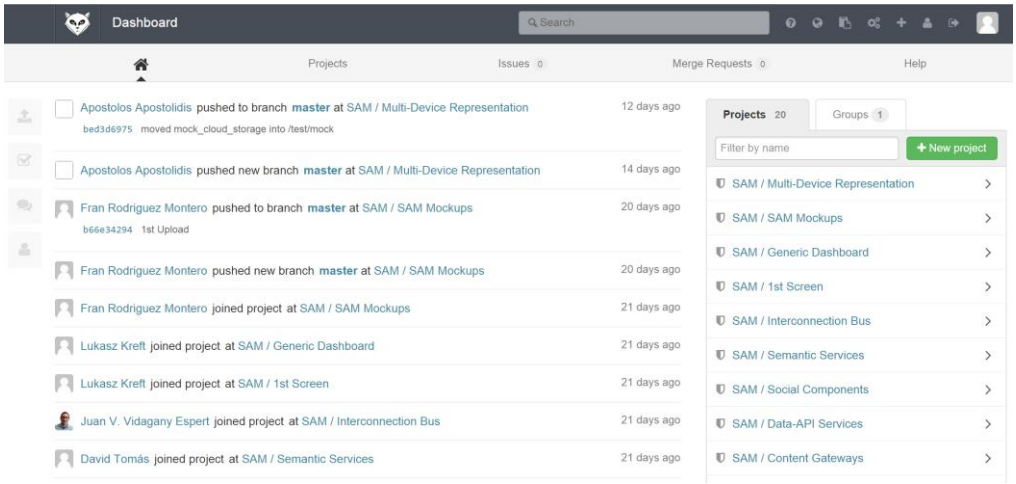

Figure 5: Source Code Management – Dashboard

3 Continuous Integration

Continuous integration (CI) is the process of combining different components to one final application (in this case, combining the SAM components to the SAM platform). The goal of this process lies in the enhancement of the software quality by applying automatic builds, integrated software tests and fast feedback. Changes in connected repositories can trigger builds of the corresponding component and by using integrated software test this CI platform can reveal errors at build time. The CI platform will be a central point which provides an overview including all components and their status viewable by all project partners. The mentioned overview will also help to recognize issues at an earlier stage.

In this section the following topics will be discussed and presented:

- Presentation of eligible candidates for the CI platform
- Selection of the CI platform
- Presentation of the current status of the CI platform
- Additional information for the project partners about the procedure

The eligible candidates will be compared using different parameters which support the selection process. The following parameters have been chosen:

Parameters	Description
Maturity & Stability	Selecting a stable and mature solution may be one of the key criteria for the selection of a CI platform.
Regularly Updated	The ICT market is developing significantly faster than any other market in the world and technologies that have not been continuously updated may be considered dead. Based on this criteria, the consortium considers it is important to choose as a base technologies that are updated regularly to reflect new market constraints.
Community	For clarifying questions and discussing possible problems of a technology, a strong community should be available.
Support of Multiple/Different SCM Systems	It is important that deployed source code management systems are supported by the selected CI platform.
Cost-Free	Describes if the solution can be used for free or the consortium has to pay for the usage.
Easy Setup and Usage	It is important that the setup can be done easily and the usage is simple but effective.

Figure 6: Continuous Integration – Parameter Description

3.1 Candidates

The following continuous integration platforms have been chosen as eligible candidates:

- Jenkins[6]
- Hudson[7]
- Travis CI[8]
- Microsoft Team Foundation Server[9]

Parameters	Jenkins	Hudson	Travis CI	Microsoft Team Foundation Server
Maturity & Stability	+++	+++	+++	+++
Regularly Updated	+++	++		+
Community	+++	++	+/-	+
Support of Multiple/Different SCM Systems	+++	+++	+++	+
Cost-Free	YES	YES	NO	NO
Easy Setup and Usage	+++	+++	N/A (Remote Host)	+

Figure 7: Continuous Integration – Parameter Overview

Microsoft Team Foundation Server is one of the options that quickly got discarded, as it is not a free product and is closely connected to Microsoft's technologies and platforms, such as the Windows operating system, and the Visual Studio IDE, which is too inflexible for a project like SAM, where other Non-Microsoft IDE's will be used.

Travis CI also falls short of the consortiums expectations, because their private builds require a paid service, although their flexible cloud based service is feature-rich and popular in the open source community.

The last two contenders are relatively similar. Both are open source deployable continuous integration servers, and Jenkins was forked from Hudson when Hudson became intellectual property of Oracle, and most of the core developers went with Jenkins, while Oracle officially supports Hudson. In the meantime, both code bases diverged, although still the same feature set exists.

3.2 Selection

For the consortium, only Hudson and Jenkins were real contenders for the continuous integration system. Due to the parameter evaluation Jenkins has been chosen over Hudson. Also in the consortium's experience, **Jenkins is much better supported** than Hudson. This also includes the open source plugins necessary for both systems. Although the features are approximately the same, for the sake of a **better-maintained and more regularly updated** system, Jenkins was chosen as CI system.

[6] http://jenkins-ci.org/
[7] http://hudson-ci.org/
[8] https://travis-ci.org/
[9] http://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx

3.3 Current Status

The continuous integration platform Jenkins has been set up by ASC and is available under **http://sam.ascora.de:8080/.**

As seen in Figure 8, for accessing the platform user credentials of a valid user account are required. At the moment all developers (RTD) have access to Jenkins and are able to create projects to build their components.

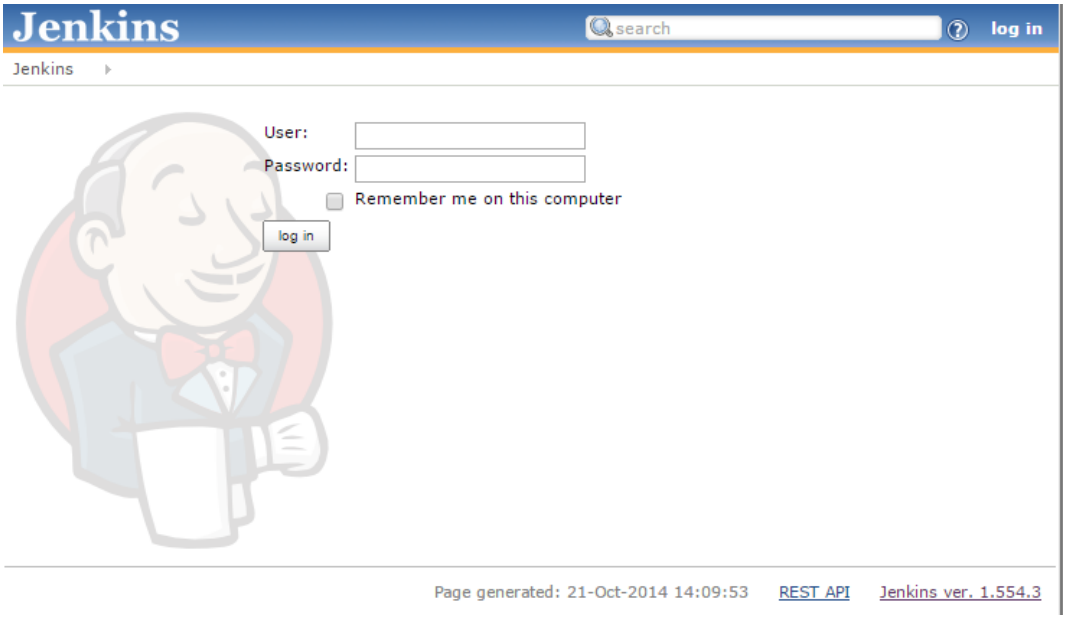

Figure 8: Continuous Integration – Login Mask

Figure 9 shows the Dashboard which provides an overview with projects and currently running builds. User can customise the view and manage the existent projects or create new ones.

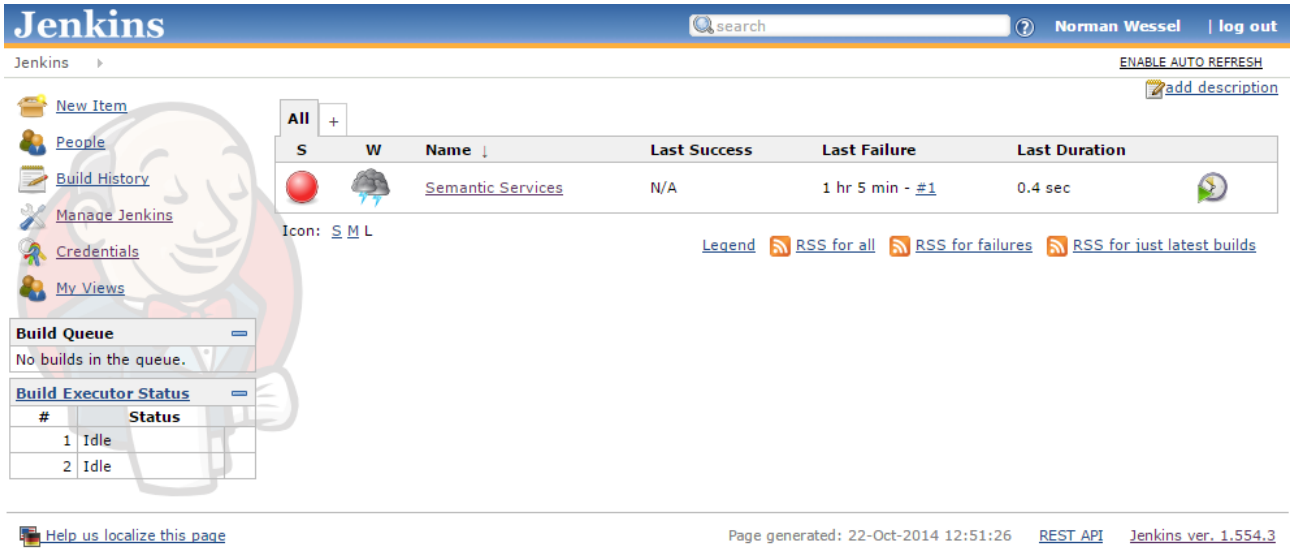

Figure 9: Continuous Integration – Dashboard

4 Issue Tracking

Issue tracking is the process of recording, tracking and finally solving issues concerning an entity. An issue can be a small bug, a new feature or an idea which relates to a SAM component. Even normal tasks can be added as an issue so the system can be used to track the implementation process. Nonetheless in all cases the issue tracking process enables to work on emerging issues in a more organizationally way, providing statistics for project/product manager and a better overview.

In this section the following topics will be discussed and presented:

- Presentation of eligible candidates for the issue tracking (ITR) system
- Selection of the ITR system
- Presentation of the current status of the ITR system
- Additional information for the project partners about the procedure

The eligible candidates will be compared using different parameters which support the selection process. The following parameters have been chosen:

Parameters	Description
Maturity & Stability	Selecting a stable and mature solution may be one of the key criteria for the selection of an ITR system.
Community	For clarifying questions and discussing possible problems of a technology, a strong community should be available.
Cost-Free	Describes if the solution can be used for free or the consortium has to pay for the usage.
Easy Setup and Usage	It is important that the setup can be done easily and the usage is simple but effective.
Development Process Support	Will the ITR system be able to support the chosen development process like SCRUM?
Issue Management	The issue management contains many different aspects, e.g. product-based and developer-based cooperation, issue overview and history. The issue management should ease the development process but should not add a higher workload.
Issue Prioritisation	To provide a better overview of responsible persons should be able to prioritize created issues. This enables more target-oriented working.

Figure 10: Issue Tracking – Parameter Description

4.1 Candidates

The following issue tracking systems have been chosen as eligible candidates:

- JIRA[10]
- Mantis[11]
- Bugzilla[12]
- Trac[13]
- GitLab[14]

Parameters	JIRA	Mantis	Bugzilla	Trac	GitLab
Maturity & Stability	+++	+++	+++	+++	+++
Community	+++	+++	+++	++	+++
Cost-Free	NO	YES	YES	YES	YES
Easy Setup and Usage	+ (only usage)	+++	++	+	++
Development Process Support	YES (Plugin)	YES (Plugin)	YES (Plugin)	YES (Plugin)	NO
Issue Management	YES	YES	YES	YES	YES
Issue Prioritisation	YES	YES	YES	YES	NO

Figure 11: Issue Tracking – Parameter Overview

Bugzilla is the great grandfather of issue tracking systems, focussing entirely on the “bug-part” using tickets. Bugzilla is rather old to be seen as a real contender for this tooling, but it has still to be mentioned as one of the default tools still used widely. Mantis and Trac are also two mixes of ticketing systems that will not be used. All these ticketing systems are too inflexible to be chosen for something complex like project management and issue tracking; they are also are very slow and feature user interfaces that reach back before the invention of the term usability.

The only real contenders for issue tracking systems are JIRA and GitLab. GitLab is directly integrated with Git and focuses on bugs, issues and features. The linkage of bugs, issues and features with commits are possible and also the assignation to the respective team members are possible. JIRA tries to be the one stop solution for everything including project management. It integrates Git, SCRUM, issue tracking, charts and task assignments.

4.2 Selection

A JIRA instance has been used as project management tool from the very beginning of this project, hosted by the project coordinator TIE. It supports all the features of the other candidates like issue management, issue prioritisation and additionally the software development methodology SCRUM (see Section 5), which will be used in this project. Due to these facts, JIRA will be used for issue tracking.

[10] https://www.atlassian.com/de/software/jira
[11] https://www.mantisbt.org/
[12] http://www.bugzilla.org/
[13] http://trac.edgewall.org/
[14] https://about.gitlab.com/

4.3 Current Status

The issue tracking system JIRA has been provided and set up by the project partner TIE. JIRA is available under **https://jira.tiekinetix.net**. The access is secured so only registered users can access the system (see Figure 12).

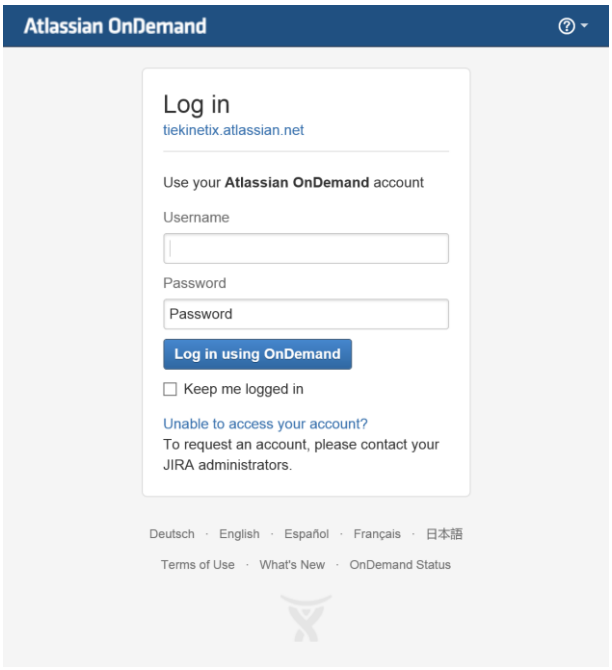

Figure 12: Issue Tracking – Login Mask

The Dashboard in Figure 13 supports different views so that each project partner can create a view fitting to its needs.

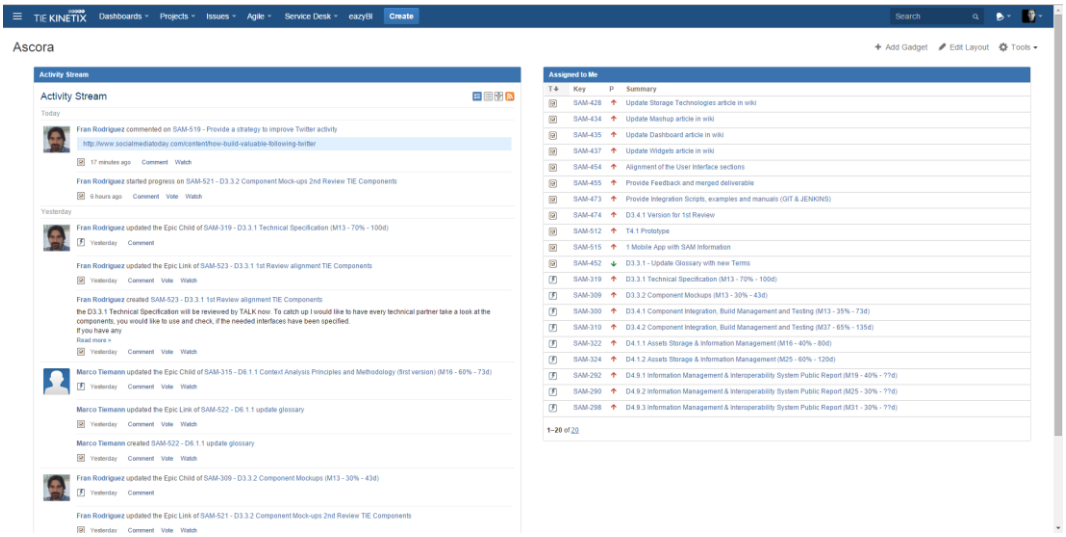

Figure 13: Issue Tracking – Dashboard

5 Software Development

Software Development is the process of developing software. For this, different development methodologies can be used. Aside from Waterfall, Prototyping, Incremental development, Spiral development, Rapid Application development and Agile development (with extreme programming and SCRUM as sub forms), a lot more flavours of software development methodologies have been invented, which are not going to be described in detail as there are a lot of differences and details that would be out of scope of this deliverable.

5.1 Selected Methodology

A mixture of Waterfall, Prototyping and SCRUM is used. Waterfall has been used until M13 requirements have been collected, and the design of the software in form of mock-ups has been done. After M13, Prototyping and SCRUM will be used in the implementation phase. The options:

- The **Waterfall**[15] model is a common design process in the software development context in which the common phases and the containing tasks are executed in a sequential manner, just like a waterfall. This process is fixed so it has to be well planned to prevent problems in later phases.
- **Prototyping**[16] is a process where prototypes of a software are created at an early stage in the project. These prototypes usually only contain a part of the features planned. The prototypes will contain more features after each iteration so at the end of the project a software version with all planned features is available.
- **SCRUM**[17] is a dynamic kind of design process where features or tasks are processed in a more agile way. To use the SCRUM process roles have to be assigned:

    - **Product Owner**, which represents the customer and provides features in form of User Stories, which then will be prioritised and added to a list, called Product Backlog. From this Product Backlog, User Stories will be used as tasks for the next implementation period, called Sprint.
    - **SCRUM Master**, which can be described as a person who helps the development team by fulfilling their tasks using SCRUM the right way and tries to prevent distracting influences.
    - **Development Team**, which will implement the specified User Stories. These User Stories will be split into smaller tasks, which can be finished in a specified interval, normally in one day. After all the User Stories should be finished within the Sprint interval.

Each of the above mentioned SCRUM roles will be covered by different partners depending on the component which has to be implemented, using the SCRUM workflow presented in Figure 14. More information on how to use SCRUM in the SAM project are to be found in Section 5.2.1.1.

[15] http://en.wikipedia.org/wiki/Waterfall_model
[16] http://en.wikipedia.org/wiki/Software_prototyping
[17] http://en.wikipedia.org/wiki/Scrum_(software_development)

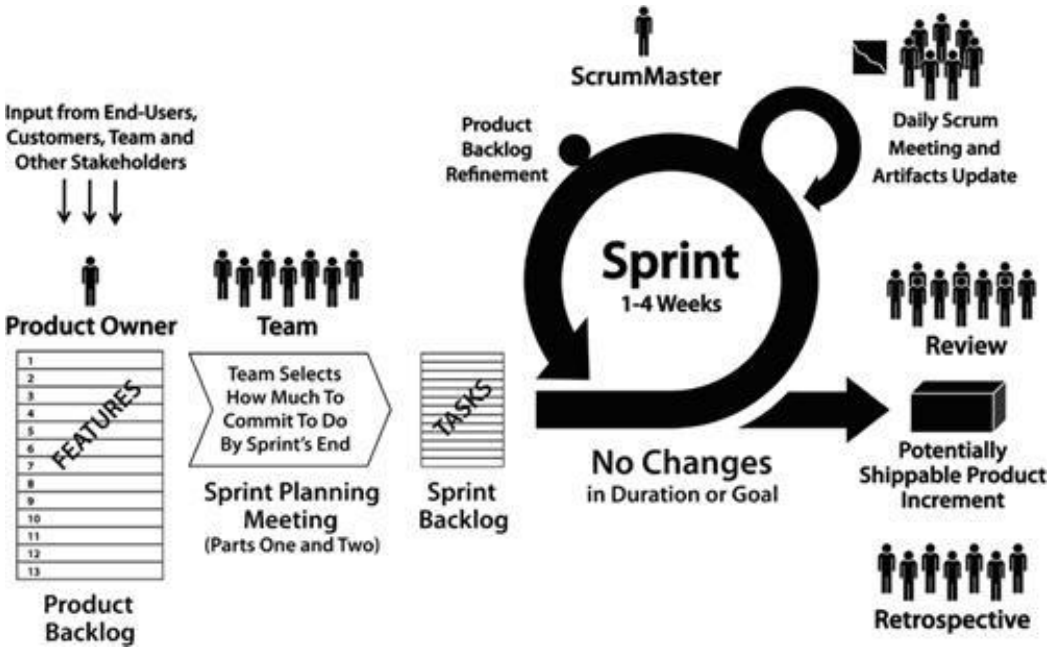

Figure 14: SCRUM Workflow in a Nutshell

As written at the start of this section, Prototyping is part of the development process as well: Two iterations of prototypes will be created to provide a running system at an early point in the project. SCRUM is an agile practice, which is very popular among managers, because it presents a system to track progress and delivers charts based on the progress recorded. Combined with JIRA it delivers a good overview and a powerful management tool as well.

5.2 Preliminary SCRUM Approach

In order to manage the SAM prototypes in WP4, WP5, WP6, WP7 (See DOW Section B1.3.1.3 Deliverables List), the consortium have decided to use SCRUM as agile methodology. Through of this approach, the WP leads and tasks leads will be able to control the progress of the developments and check if the SAM requirements defined in D2.3 User Stories and Requirements Analysis are being implemented correctly. Besides, in any moment of the development, any non-technical partner can see the progress of the prototypes and provide their feedback about it. This way to work will benefit better final results in the developments.

In the following sections it is explained how the SCRUM methodology works, the different concepts related to it such as roles or sprints and how will be the procedure from the WP Lead and Task Lead point of view.

5.2.1 SCRUM Methodology

SCRUM[18] is an iterative and incremental agile software development framework for managing product development. It defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal", challenges assumptions of the "traditional, sequential approach" to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of

[18] https://www.scrum.org/

all team members, as well as daily face-to-face communication among all team members and disciplines in the project.

The most important SCRUM concepts related to SAM will be explained in the following sections.

5.2.1.1 Roles

In SAM project, the SCRUM roles will be the following:

- The **Product Owner** determines what needs should be built. In SAM, the product owner will be the coordinator
- The **SCRUM Master** ensures this process happens as smoothly as possible, and continually helps to improve the process, the team and the prototype. In SAM, the SCRUM master will be each WP Lead, who will be in charge of controlling the progress of the different prototypes based on the User Stories and tasks created by the tasks leads.
- The **Development Team** build what is needed, and then demonstrate what they have built. Based on this demonstration, the Product Owner determines what to build next. In SAM, the team will the partner team in charge of develop each prototype.

5.2.1.2 Sprints and Backlog

A **Backlog** is a list of the outstanding User Stories, tasks and features for a product (Product Backlog) or Sprint (Sprint Backlog).

Sprints are used to describe periods of planned work that facilitate internal deadlines. In SAM, these Sprints will be done in a monthly interval so that there is enough flexibility in subtask scheduling and planning. Due to the fact that project partners are located in different locations the oftentimes-mentioned concepts described with “Stand-up meeting” or “SCRUM meeting” cannot be made traditionally, so these meetings will be performed using Skype or similar technologies.

These Sprints will start with a so-called **Sprint Planning Meeting** where the development team will commit to specific tasks chosen by themselves. Each Sprint will produce a release which will be reviewed at the end in the **Sprint Review Meeting**, where Product Owner and the development team will evaluate the results together and discuss any problems. After this, the Sprint backlog will be filled again during the next **Sprint Planning Meeting**.

5.2.2 SAM SCRUM Approach

In order to implement the SCRUM methodology, the JIRA tool will be used. A tutorial about how to use this tool and more information about SCRUM methodology can be founded it in Dropbox folder “\SAM\WPs\WP1\D1.2\Jira help”. However the most important JIRA concepts and the SAM SCRUM procedure will be detailed in the following subsections.

5.2.2.1 JIRA Concepts

JIRA has a wide documentation[19] to know all concepts and definitions. This section aims to explain the most important concepts to know in SAM SCRUM Approach. These are the different kinds of issues (Epic, User Story and Task) and how to connect them. The definition of the different kind of issues is the following:

[19] https://confluence.atlassian.com/display/AGILE/Working+with+Issues

- **Epic**: An Epic captures a large body of work. It is essentially a large User Story that can be broken down into a number of smaller stories. It may take several Sprints to complete an Epic. Epics provide you with an additional hierarchy of story management, providing planning guidance for groups of issues within, or across, projects. This allows Scrum Masters and Product Managers to measure important groups of issues that are all related by a common theme. For the sake of SAM an Epic is identified with a Deliverable.
- **Story**: A story or User Story is a software system requirement that is expressed in a few short sentences, ideally using non-technical language. In JIRA Agile, a story is represented as a JIRA Story, and individual tasks within the story are represented as sub tasks. A Story will be defined as a User Story (see D2.3) that will be decomposed in further Stories and tasks affordable to be implemented in a specific Sprint.
- **Task**: A task is a unit of work contained within a story. In JIRA Agile, individual tasks are represented as sub-task issues, and stories are represented as parent

To connect them, it is important to know how to connect User Stories with Epics and to link tasks with User Stories.

- **Assign an Epic in a User Story:** When a new User Story, in the form, should be assigned the Epic in the box Epic Link. It is possible to do that editing the User Story (See Figure 15).

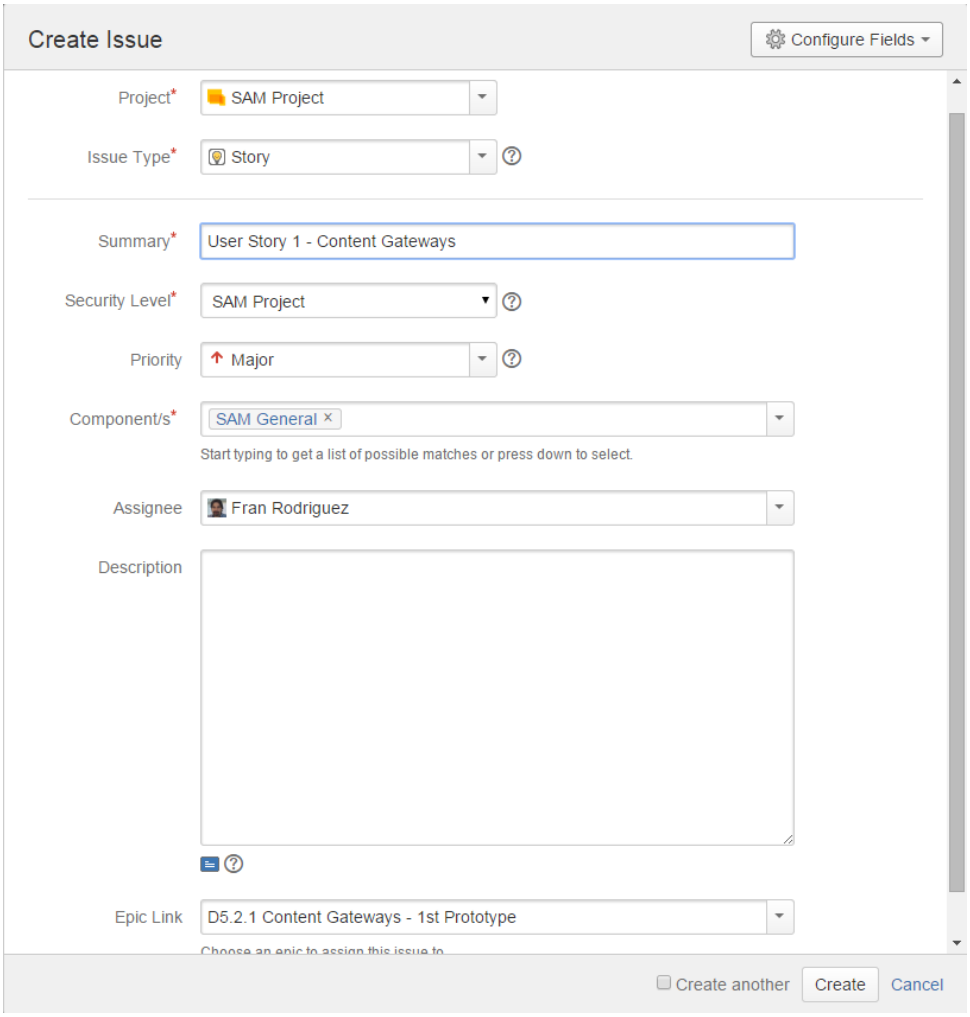

Figure 15: Assign Epic Link in a New User Story

- **Link Tasks with User Stories:** Once a User Story and the different related tasks have been created, these can be linked. The task lead should edit the User Story, goes to the “More” menu and select the “Link” option. In the next window should be selected the related issue to be linked (See the following screen shots).

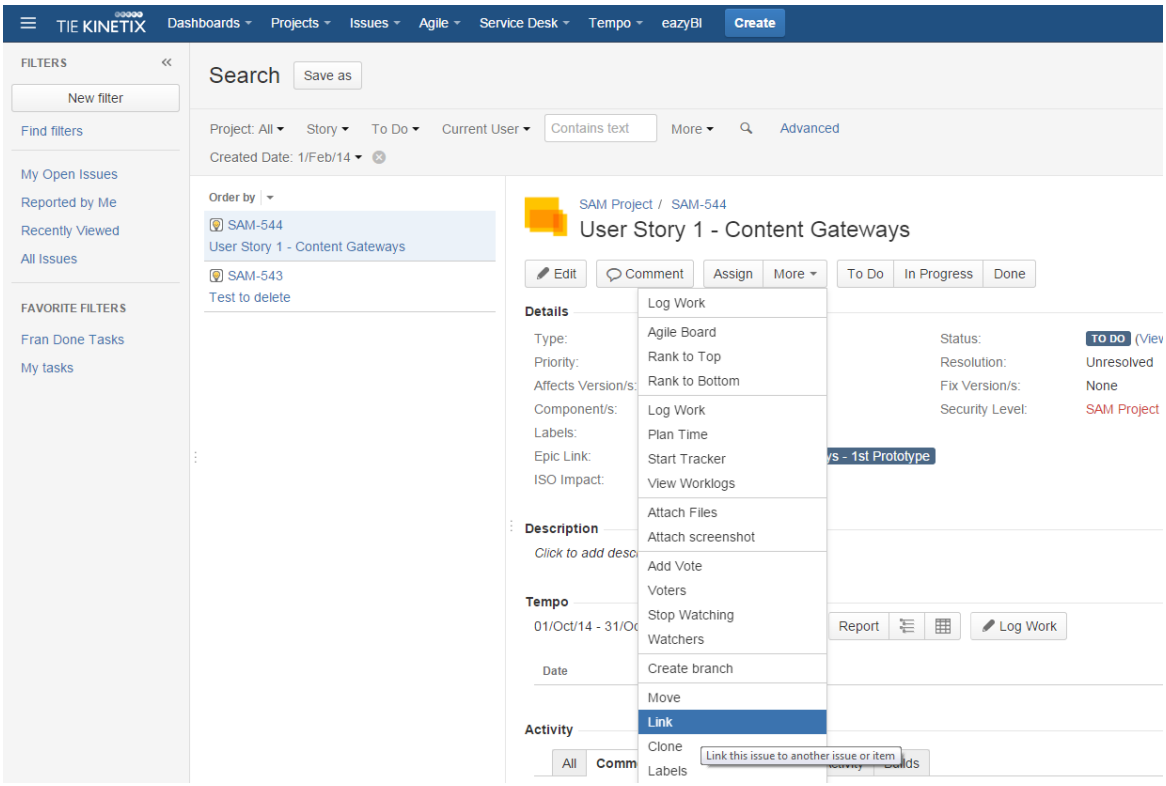

Figure 16: Link Tasks in User Story – Step 1

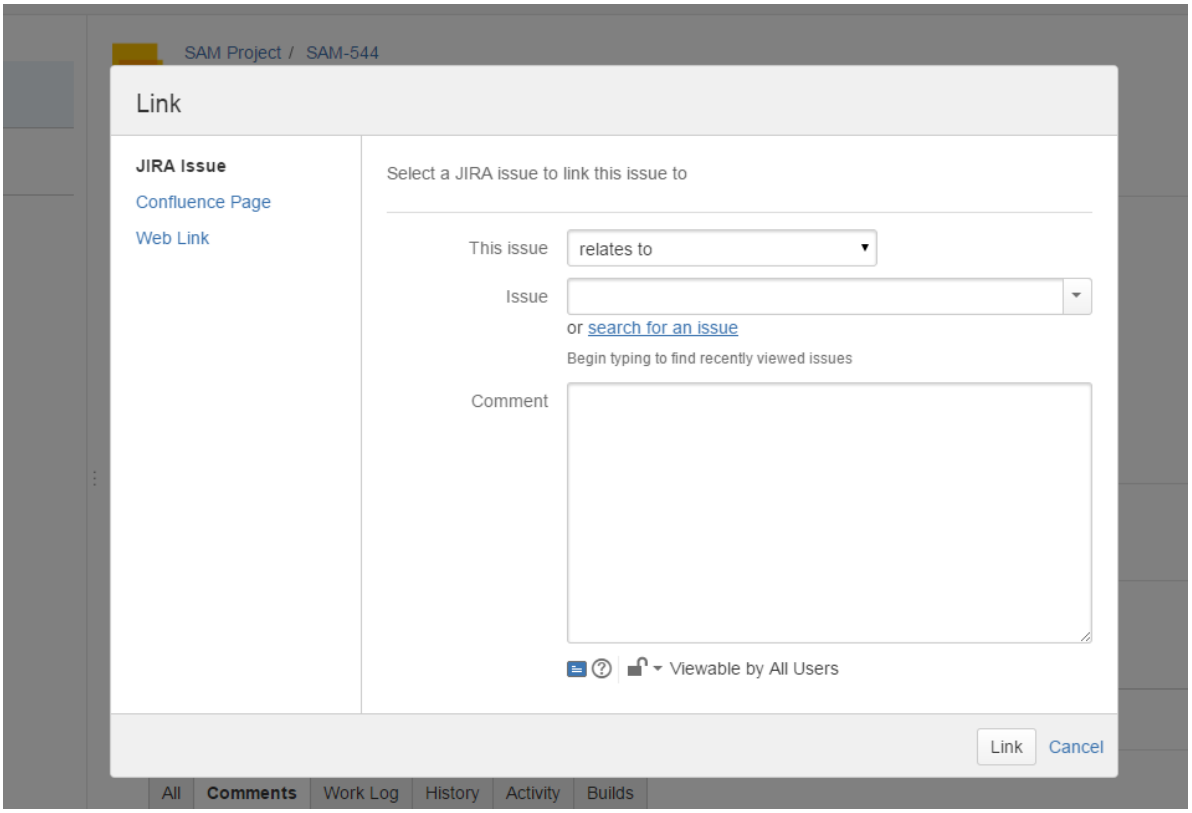

Figure 17: Link Tasks in User Story – Step 2

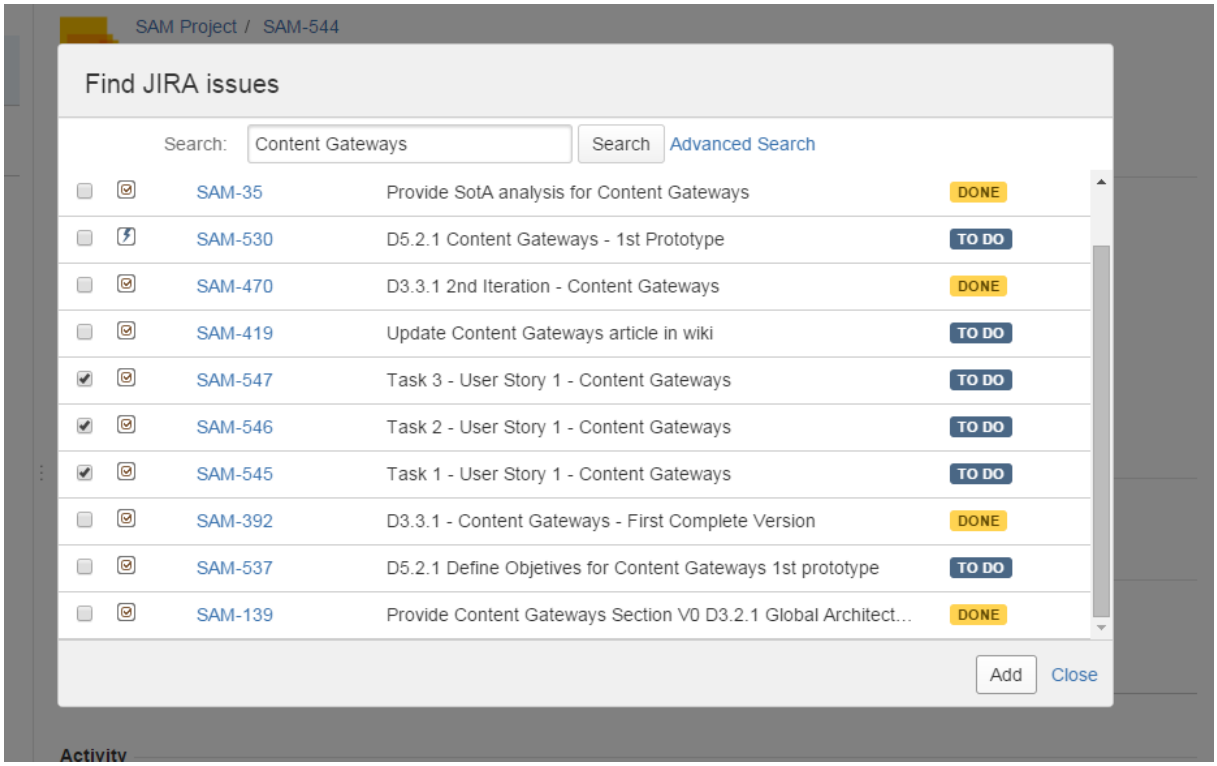

Figure 18: Link Tasks in User Story – Step 3

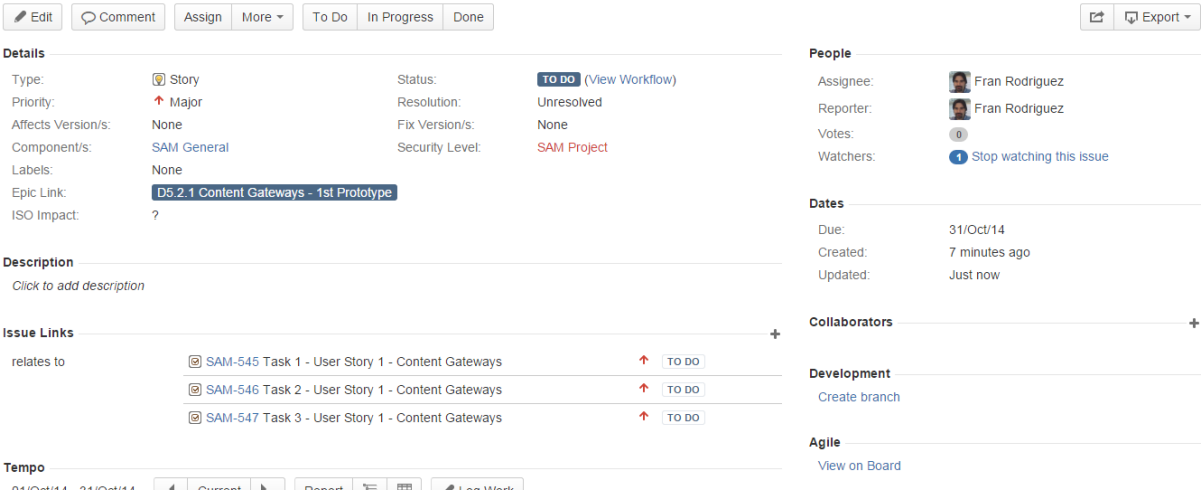

Figure 19: Link Tasks in User Story – Step 4

The hierarchical structure in SAM IS the following:

- **Epics:** Each Epic represents a deliverable, independent of its Nature
- **User Stories:** Each User Story represents a User Story defined in D2.3
- **Tasks:** Each task represents a necessary task to carry out a User Story

5.2.2.2 SAM SCRUM Procedure

The procedure is:

- The WP Lead should create an Epic for each deliverable. For instance, in WP5, the task 5.2 Content Gateways has three milestones: M19, M25 and M31. So the WP Lead should also create three Epics (e.g. D5.2.1 Content Gateways – 1st Prototype).
- The Task Leads should create one User Story for each User Story defined in the D2.3 deliverable.
- The Tasks Lead will break down each User Story in different stories or tasks and will link these tasks to the User Stories (see Figure 16 to Figure 19)
- In each Sprint Planning meeting, the WP Lead will move the tasks from the Backlog to the suitable Sprint after discussion with Tasks Leads
- In each Sprint Review meeting the Task Lead will show the progress of the task with a video demo. This demo will be added to the related JIRA task

5.2.2.3 General Agreements about Sprints

- The initial idea is that each Sprint has duration of four weeks (simplified to one month). The Sprint starts with the Sprint Planning meeting in the 1st week and ends with the Sprint Review meeting in the last week. In order to avoid mixing the Sprint Planning meeting and the Sprint Review meeting, the Sprint Planning meeting will be conducted one day after the Sprint Review meeting. This duration can be adapted for any WP Lead.
- Example for Work Package 5:

    - **Sprint 1 Planning Meeting:** Friday 7th of November
    - **Sprint 1 Review Meeting:** Thursday 27th of November
    - **Sprint 2 Planning Meeting:** Friday 28th of November

6 Integration of Technical Components

This section will cover the integration process of the SAM platform. First, the integration process and its project-internal management will be defined (see Section 6.1) and then the outcomes will be described (see Section **Error! Reference source not found.**)

6.1 Definition and Management

Integration describes the composition process of multiple entities resulting in one entity. In SAM the SAM platform will consist of different components as to be seen in Figure 20. To provide a fully functional system the components have to be merged together in a way which ensures the functionality of each component and the function of the system as a whole. This goal will be achieved with the combination of the different infrastructure systems described in this document.

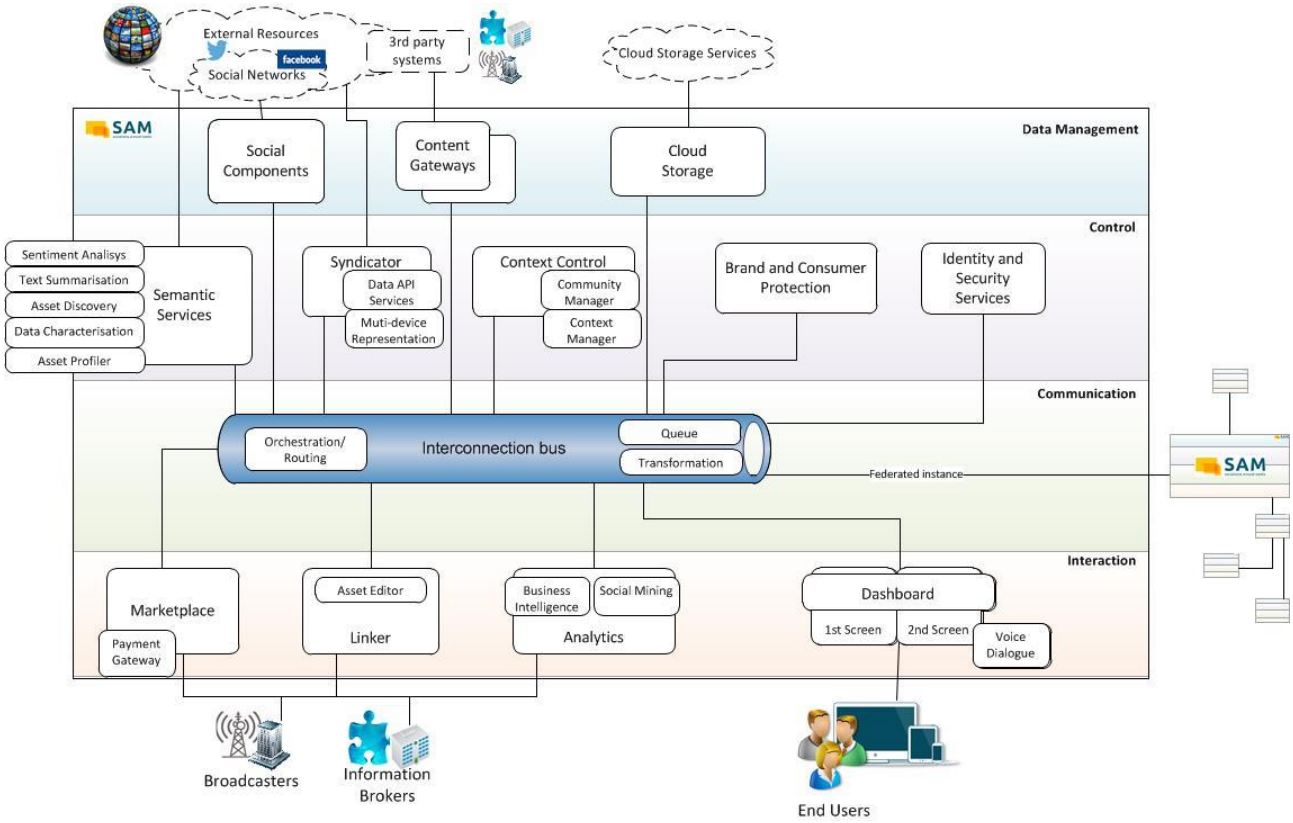

Figure 20: SAM High Level Architecture

The CI platform Jenkins will create deployable entities (e.g. web applications, services, etc…) using the repositories provided by the SCM system Git. Each entity will represent a SAM component providing required functionalities. These entities will be deployed in an automatic way to provide the SAM platform functionality. Due to the fact that the SAM project will also provide prototypical implementations for mobile devices (2nd Screen) and Smart TV's (1st Screen), the integration process will produce three different entities:

- **SAM platform**, representing the server side logic and providing graphical user interfaces for the different stakeholders
- **1st Screen**, representing the SAM client application running on Android Smart TV's

- **2nd Screen**, representing the SAM client application running on Android mobile devices

6.2 Outcomes

This section will be updated in M37 and will include:

- The final integration status of the SAM platform and its supporting entities
- Any difficulties which arose and had to overcome
- Based on the difficulties decisions made and lessons learned

7 Testing

This section will cover the software testing process of the SAM platform. First, the testing process and its project-internal management will be defined (see Section 7.1) and then the outcomes will be described (see Section **Error! Reference source not found.**)

7.1 Definition and Management

Software testing is the process of testing and evaluating software using one or more testing methods. The goal of software testing is to check if the software fulfils existing requirements and to measure the software quality. Tests can be conducted manually or automatically, by software or by humans.

To guarantee a high software quality level in the SAM project the following test methods are planned (see also Figure 21):

- **Unit Tests:** For each SAM component unit tests should be provided, if feasible. Unit tests will evaluate the functionality on component level. The unit tests should be performed automatically when a Jenkins build is triggered, so that developers can react fast on any issue arising (more information on continuous integration, see Section 3). Each Task Lead will be responsible to provide unit tests.
- **Integration Tests:** To evaluate the interaction between different SAM components, integration tests should be provided. This enables the evaluation of the SAM platform on a system-wide level. The integration tests should be provided by the WP Lead in cooperation with the different Task Leads. Additionally the integration tests should be automated and, if possible, executed during the building stage.
- **Quality Assurance Team:** The task of this team, which will consist of the user partners, will be conducting user acceptance tests, which do not try to reveal code issues, but checks if the SAM platform as a whole is acceptable regarding the stakeholder's use case specified in the DOW as tasks T8.2 and T8.3. Especially the opinions of the End User will be of importance. The user acceptance tests should be conducted and repeated for each finished software prototype.

Figure 21: SAM Test Methods

Each issue should be reported using the ITR system chosen in Section 4.

7.2 Outcomes

This section will be updated in M37 and will provide the following information:

- The final testing status of the SAM platform and its supporting entities
- Any difficulties which arose and had to overcome
- Based on the difficulties decisions made and lessons learned

Annex A: Additional Information

This section will contain tutorials and other helpful information regarding the following sections:

- **Source Code Management:** This section will provide a manual to work with a Git repository
- **Continuous Integration:** This section will provide a manual to compile a usable version based on source code contained by one or more Git repositories
- **Issue Tracking:** This section will describe how to create an issue using the JIRA platform

A1 Source Code Management: How to Use Git

Git will store the source code of our components in so called repositories. To access these repositories the user has to have a Secure Shell (SSH) key pair and a working GitLab account. In Section A1.1 the creation and use of this SSH key will be explained. Then in Section A1.2 the workflow will be presented.

A1.1 Create and Use SSH Key Pair

The following section will act as a guide for creating a SSH key pair.

A1.1.1 Linux

To generate a new SSH key just open your terminal and use the code below. The ssh-keygen command prompts you for a location and filename to store the key pair and for a password. When prompted for the location and filename press enter to use the default. It is a best practice to use a password for an SSH key but it is not required and you can skip creating a password by pressing enter. Note that the password you choose here can't be altered or retrieved.

ssh-keygen -t rsa -C "youremail@adress.com"

Use the code below to show your public key.

cat ~/.ssh/id_rsa.pub

Now log into GitLab and click on Profile Settings in the upper right corner (see Figure 22). After that click on the SSH Keys section and click on the “Add SSH Key” button. Now copy-paste the key in the terminal to the 'My SSH Keys' section (see Figure 23) under the 'SSH' tab in your user profile. Please copy the complete key starting with ssh- and ending with your username and host.

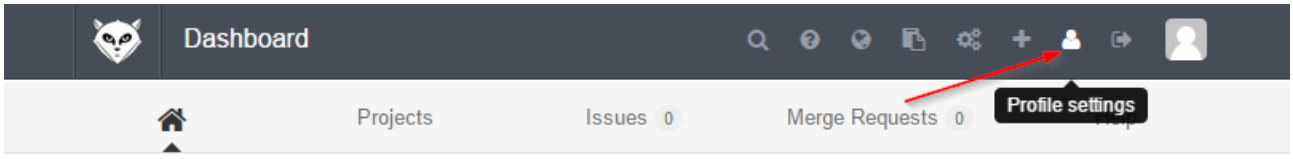

Figure 22: GitLab – Profile Settings

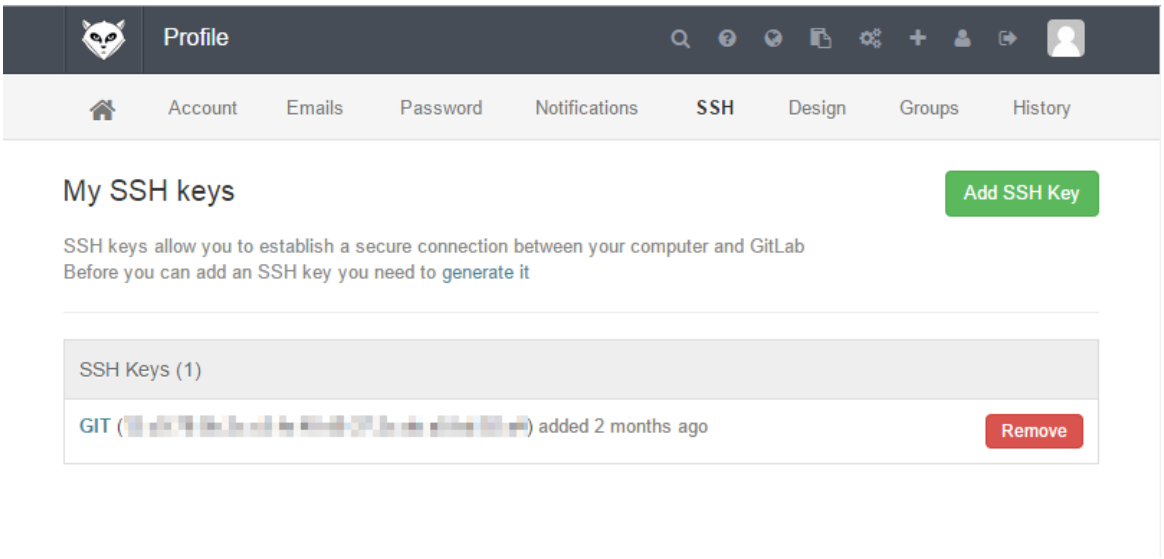

Figure 23: GitLab – SSH Section

Also the following link to the video tutorial explains this method in an easy way:

https://www.youtube.com/watch?v=54mxyLo3Mqk

A1.1.2 Windows

To generate a new SSH key you have to download the tool "Putty Key Generator"[20]. After downloading start the tool and you will see the GUI shown in Figure 24.

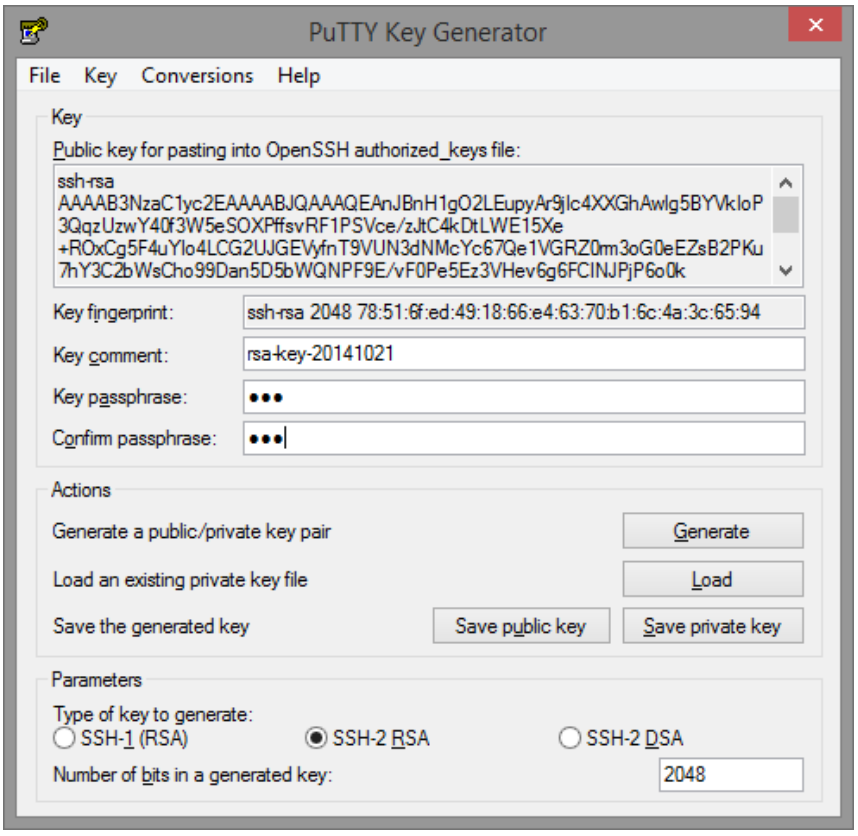

Figure 24: Putty Key Generator GUI

[20] http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

Click on Generate and move the cursor over the window until the key has been generated. Then copy the OpenSSH key which is located in the text field, above the Key fingerprint input. Now log into GitLab and click on Profile Settings in the upper right corner (see Figure 22). After that please click on the SSH Keys section (see Figure 23) and click on the “Add SSH Key” button. Now copy-paste the key to the 'My SSH Keys' section under the 'SSH' tab in your user profile. Please copy the complete key starting with ssh- and ending with your username and host.

Now please save both keys (public and private) to your disk. **Keep the private key private!**

A1.2 Workflow

This section will explain how to make a local copy of the repository so the user can edit the code and then commit it to the original repository. Due to the multiplicity of usable tools to clone a repository, this tutorial will be based on the console commands necessary only. Other tools may provide graphical user interfaces but under the covers they use the same notations for commands which can be found in the next steps.

The following steps have to be conducted:

1. **Clone a repository:** Use the following command to retrieve the repository. This has only to be done at the first time. The specific address for a component, i.e. **git@sam.ascora.de:sam/sam-mockups.git**, can be found in the component overview in the GitLab:

   git clone git@sam.ascora.de:sam/componentname.git

2. **Checkout a branch:** Most repositories will use only one branch, the Master branch. Use the following command to retrieve the master branch:

   git checkout -b master

3. **Commit code:** After editing the code the changes have to be committed back to the repository. There are two steps required. First the user has to commit the code to the local repository with a meaningful commit message. This can be done by the following command:

   git commit -am "This is my commit message"

4. **Push commits:** In the second step the user will upload the commits to the remote repository in the GitLab, which can be done by the following command:

   git push origin master

A2 Continuous Integration: How to Use Jenkins

Jenkins offers the triggered compilation and provision of components based on projects. A project is a task which compiles the source code provided by a Git repository and finally provides an artefact, which is a file which is usable for further deployment or usage. Projects can be triggered by code changes in a repository so the owner of the component repository is able to verify the correctness of the provided code.

A2.1 Create a Project

To create a project, an already existing project can be used as a template. First click on the “New Item” link (see Figure 25). Then provide the item name, which should be the

same as the GitLab project name to prevent confusion. At last select “Copy existing item” to choose a fitting project to copy, and click on the “OK” button.

In the next screen the user has to customize the Git repository and the build process, so the desired artefact will be created and stored (see Figure 26).

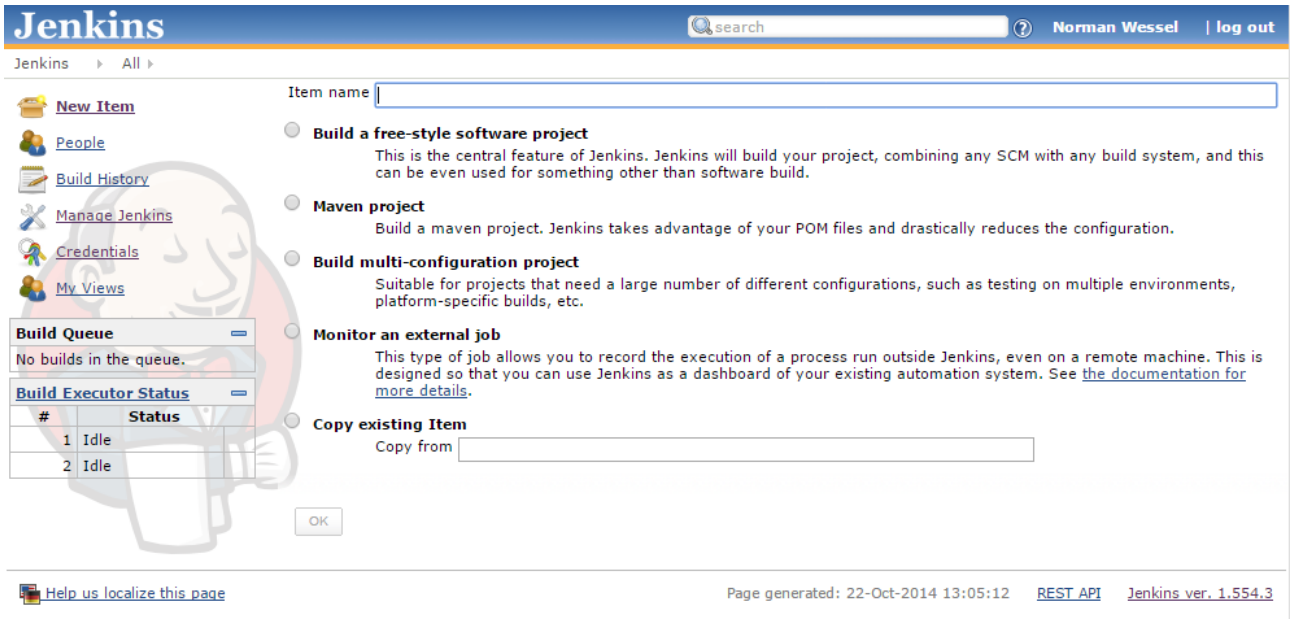

Figure 25: Continuous Integration – Create a Project

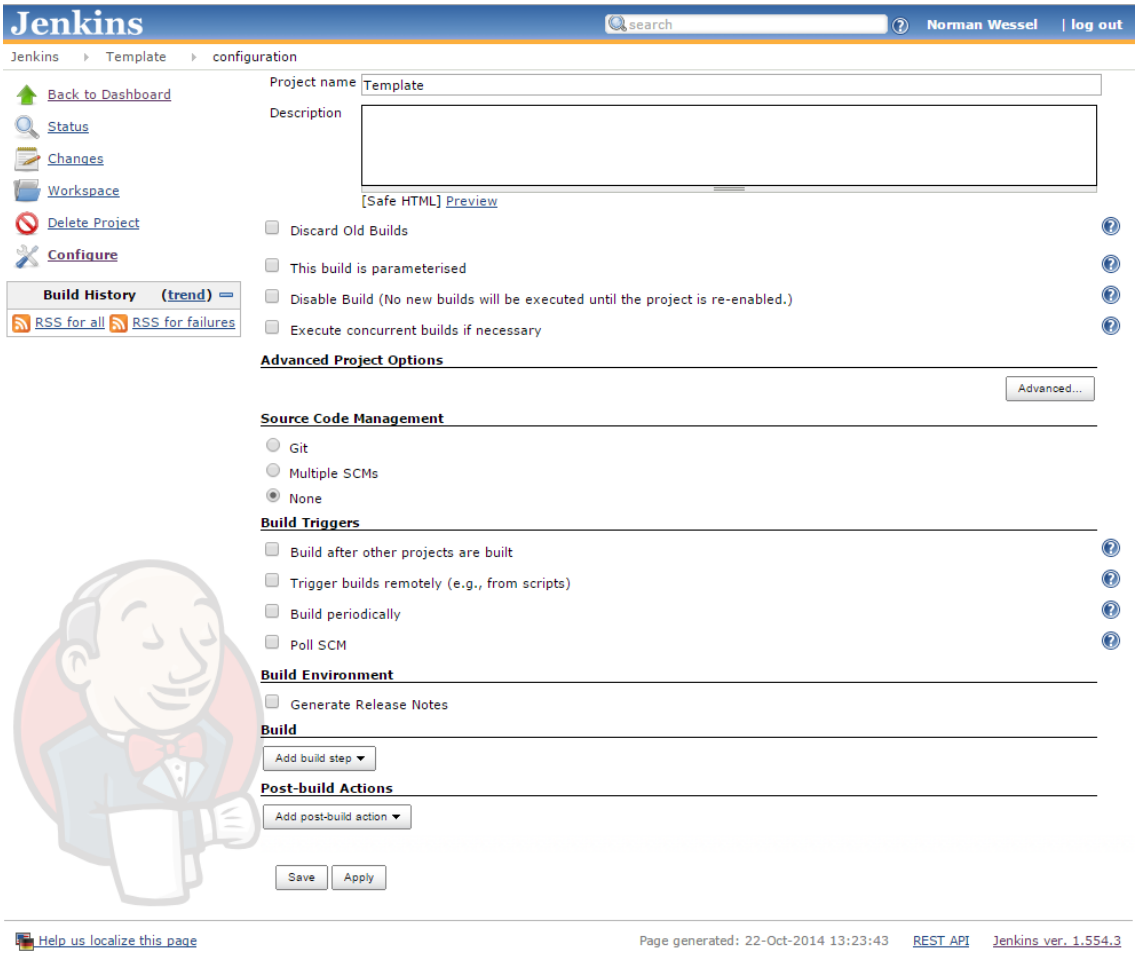

Figure 26: Continuous Integration – Customise Project

A2.2 Retrieving Artefact

To retrieve an artefact, the user has to start or schedule a build. Jenkins offers this function at different locations in the application but the user has to be logged in to use them. The quickest way to start a build is to use the “Schedule a build” button (green arrow) in the Dashboard (see Figure 9 on the right side of the project column). Other possibilities can be configured for a job, for example to always build on a changed codebase in the SCM system.

When the build was successful, the artefacts will be stored and can be retrieved by the user on the project’s site.

A3 Issue Tracking: How to Use JIRA

As there is already an existing tutorial for JIRA which explains its functionality in a detailed way, the provision of a new tutorial is not necessary. The tutorial is internally available in Dropbox which has been chosen by the consortium for storing documents related to the project.

Annex B: List of Provided Systems

System	URL
Source Code Management	http:/sam.ascora.de
Continuous Integration	http:/sam.ascora.de:8080
Issue Tracking	https://jira.tiekinetix.net

References