Platform Release Plan

November 30, 2015

Deliverable Code: D7.1

Version: 1.0 – Final

Dissemination level: Public

This deliverable describes the initial plan for software engineering infrastructures, tools and processes that will be employed to manage releases of software components that make up the OpenMinTeD platform.

H2020-EINFRA-2014-2015 / H2020-EINFRA-2014-2
Topic: EINFRA-1-2014
Managing, preserving and computing with big research data
Research & Innovation action
Grant Agreement 654021

Document Description

D7.1 – Platform Release Plan

WP7: Platform Integration, Testing and Deployment	
WP participating organizations: USFD, ARC, UNIMAN, UKP-TUDA, INRA, AK, GRNET.	
Contractual Delivery Date: 11/2015	**Actual Delivery Date:** 11/2015
Nature: Report	**Version:** 1.0 (Final)
Public Deliverable	

Preparation slip

	Name	Organization	Date
From	Ian Roberts Richard Eckart de Castilho	USFD UKP-TUDA	16/11/2015
Edited by	Ian Roberts	USFD	26/11/2015
Reviewed by	Theodoros Manouilidis Thomas Margoni	ARC Univ. of Stirling	25/11/2015 28/11/2015
Approved by	Theodoros Manouilidis	ARC	30/11/2015
For delivery	Mike Chatzopoulos	ARC	30/11/2015

Document change record

Issue	Item	Reason for Change	Author	Organization
V0.1	Draft	First full draft assembled from forum discussions	Ian Roberts	USFD
V0.2	Draft	Additions following 19/11 discussion	Ian Roberts	USFD
V1.0	Final	Merged reviewer edits	Ian Roberts	USFD

1. Table of Contents

Disclaimer

This document contains description of the OpenMinTeD project findings, work and products. Certain parts of it might be under partner Intellectual Property Right (IPR) rules so, prior to using its content please contact the consortium head for approval.

In case you believe that this document harms in any way IPR held by you as a person or as a representative of an entity, please do notify us immediately.

The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document hold any sort of responsibility that might occur as a result of using its content.

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of the OpenMinTeD consortium and can in no way be taken to reflect the views of the European Union.

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 28 Member States of the Union. It is based on the European Communities and the member states cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors. (http://europa.eu/)

OpenMinTeD is a project funded by the European Union (Grant Agreement No 654021).

Publishable Summary

This deliverable describes the tools, infrastructures and processes that will be followed when developing and releasing OpenMinTeD platform components. The initial plan is built around the following broad principles:

- Open by default – adopt FLOSS (Free/Libre Open Source Software) licences by default unless there are strong reasons to do otherwise, and carry out development activity in public as far as possible.
- Test-driven development – ensure there are comprehensive unit/integration/functional test suites for each component as appropriate.
- Continuous integration – ensure that all modules are built and tested regularly so build errors can be spotted and corrected as soon as possible.
- Leverage existing best practice – where possible, make use of existing best practice and established tools rather than re-inventing our own.

The project will make use of GitHub for source code management and issue tracking (https://github.com/openminted), Jenkins for continuous integration (http://builds.openminted.eu), and a release management process inspired by that used in the Apache Software Foundation to ensure high quality software releases. However we also recognize that requirements may change during the lifetime of the project, and the plan is designed to be flexible and adaptable as appropriate.

1. Introduction

The OpenMinTeD project will involve the release of a variety of different software components, developed by different partners in the project and third parties, written in various programming languages, with different sets of library dependencies, etc. and a robust set of software engineering tools and processes will be required to manage these releases appropriately. This deliverable describes the initial plan for release management in OpenMinTeD, but recognises that the project must retain the flexibility to adjust the plan during the course of the project (and beyond) in light of new tools or best practices becoming available. The initial plan is built around the following broad principles:

- Open by default – adopt FLOSS licences by default unless there are strong reasons to do otherwise, and carry out development activity in public as far as possible.
- Test-driven development – ensure there are comprehensive unit/integration/functional test suites for each component as appropriate.
- Continuous integration – ensure that all modules are built and tested regularly so build errors can be spotted and corrected as soon as possible.
- Leverage existing best practice – where possible, make use of existing best practice and established tools rather than re-inventing our own.

Section 2 discusses the tools and infrastructure that we intend to use for the development and release of OpenMinTeD components, section 3 outlines the *process* that modules will go through when producing releases, and section 4 gives a first snapshot of the timeline for releases of platform components – this timeline should be seen as a living document and will be updated as the project evolves.

2. *Software Engineering Infrastructure*

With the guiding principles in mind, the OpenMinTeD project intends to standardise on a core set of established software engineering tools and infrastructure for modules developed by the project.

2.1 Source code management and issue tracking

Source code for OpenMinTeD modules will be placed under a suitable FLOSS licence and hosted as public projects on the popular *GitHub* platform[1], unless the modules have dependencies on third party components or data released under licences that preclude this. A GitHub *organization* name has been reserved for core platform modules, at https://github.com/openminted. GitHub was chosen for a number of reasons, principally because it is already a recognised and respected name in the FLOSS community, and its fork-and-pull development model makes it straightforward to manage third party contributions while maintaining overall editorial control.

Every GitHub project comes with its own *issue tracker*, where users can file bug reports and feature requests, and OpenMinTeD will use this as the main venue for interaction with users and other developers. Third parties who wish to contribute to a GitHub hosted project do so by creating their own personal *fork* of the original project, making their changes in that project, then offering the changes back to the upstream project via GitHub's *pull request* mechanism. The owner of the upstream repository has the option to review the offered changes and decide whether to accept them as-is, accept them with modifications or reject them entirely.

We envisage that in the first instance the core OpenMinTeD project developers responsible for a particular module will have direct access to push their changes to the main project, and contributions from third parties will be managed through the pull request mechanism. Outside contributors with a track record of good quality contributions may later be granted direct push access.

For modules where the open development methodology of GitHub is not appropriate (for example modules that rely on tools or data with more restrictive licence conditions), there will be the option to create a private issue tracker in the OpenMinTeD redmine[2] instance. This may also prove necessary for otherwise-open modules where specific user issues can only be replicated using proprietary data. But in general there is a presumption towards using the public tools on GitHub wherever possible.

2.2 Continuous Integration (CI)

The use of continuous integration is well established in open source software projects as a good way to catch potential problems early, while they still require minimal effort to fix. OpenMinTeD will set up a

[1] https://github.com

[2] http://www.redmine.org

Jenkins CI server[1], hosted at http://builds.openminted.eu, and project members will be able to log in to this server using their common project-wide username and password. Each module will have (at least) one Jenkins configuration to build and test its "master" branch whenever changes are pushed. The master branch must build successfully and pass all its tests at all times – significant refactoring of code should be performed on a different branch and merged to master only once it reaches a state where the Jenkins build will succeed. The Jenkins build will also be set up to check non-functional requirements such as the presence of appropriate licence headers on all source code files.

For projects that involve platform-specific native code modules, consideration may be given to running the Jenkins build on several different platforms (e.g. a representative GNU/Linux distribution, Microsoft Windows and possibly Mac OS X).

As modules develop it may be necessary to maintain multiple versions in different branches (e.g. master for the latest new developments in version 2, and a maintenance branch for fixes to version 1). If appropriate, such maintenance branches may have their own associated Jenkins build configurations.

2.3 Release Artifacts

Whenever a release "tag" is pushed to a GitHub repository, GitHub automatically makes available a snapshot bundle of the state of the source tree at the point of that release. Additional files can be added to the release, for example precompiled binaries, a WAR file for a Java web application, etc. and OpenMinTeD will make use of this mechanism to publish releases of developed modules.

However, for certain programming languages there are well-established routes that already exist for distributing release artifacts. For Java and other JVM languages such as Groovy and Scala, compiled JAR files are typically published to the *Maven Central Repository*, the default location from which many Java build tools (Apache Maven itself, but also Gradle, Apache Ivy, and others) can fetch dependencies. OpenMinTeD will follow these conventions where they exist, in particular Java snapshot artifacts will be staged to an OpenMinTeD Maven repository (http://repo.openminted.eu) and release artifacts will be staged to the Sonatype OSS repository[2], which feeds them into Maven Central. We expect that Java-based modules produced by the project will themselves be built using Maven or Gradle and will take their dependencies from Maven Central.

Developers should consider publishing Docker images[3] for module releases to allow the components to run in a stable, predictable environment. This is particularly relevant for larger and more complex

1 https://jenkins-ci.org

2 http://www.sonatype.org

3 https://www.docker.com

modules that involve co-operating components in different programming languages. A Docker Hub organization namespace “openminted” has been created for use by the project.

3. *Release Management Process*

When developing a large number of related and interdependent modules, each with their own release cycle, it is important to have a robust process in place to ensure the quality of the resulting releases. The following process is recommended for all OpenMinTeD modules, but can be varied by the team responsible for a particular module by their agreement.

3.1 Version numbering

OpenMinTeD modules will use semantic version numbers[1] of the form major.minor.patch, with a "-SNAPSHOT" suffix for development versions (this is a Maven convention, appropriate conventions should be followed for non-Java modules). In semantic versioning, the major version number is incremented when there are backwards-incompatible changes to the public API, the minor version when new features are added in a backwards-compatible manner, and the patch version when backwards-compatible bug fixes are made that do not add new features. Note that if a bug fix breaks backwards compatibility in the public API then this causes a *major* version change. A new module starts at version 0.0.1-SNAPSHOT, and the first public release with a stable API should be version 1.0.0.

3.2 When to release?

For each principal group of OpenMinTeD components there are three major milestones laid out in the initial release plan (see section 4), and individual components should aim to have a coherent and fully-working release ready for these milestones. The key word is *coherent* – if it is necessary to meet a specific release date then incomplete features may need to be dropped entirely rather than being released half-finished. Additional interim releases may also be made as appropriate, e.g. when specific requirements are met or key features implemented.

3.3 Roles and responsibilities

When it has been decided to make a release of a specific component, a *release manager* will be appointed by the developers of the component, who is responsible for shepherding the release through the process. If there are no volunteers for this role for a particular release, the relevant WP leader will nominate an appropriate person. They do not necessarily have to be the person who actually performs every step, but they are responsible for ensuring the process is followed, delegating to others as

[1] http://semver.org

required. The other developers involved with the module will need to review and approve the release packages before they are released to the public.

The release manager will also be responsible for the reporting of the release status in the OpenMinTeD Redmine, used for the overall project management. This person should be able to provide regular updates on the status of the components and features of each release, at a higher level of abstraction than the one provided by the issue tracker on GitHub, grouping this information for reporting purposes.

3.4 Feature freeze

When all the features required for the release have been implemented, the release manager will declare a feature freeze. At this point they will create a release *branch* in the appropriate GitHub repository (or repositories), and renumber the "master" branch to a new snapshot version – if master is at version 1.0.2-SNAPSHOT before the freeze then it should be changed to 1.0.3-SNAPSHOT. New features can now be added on the master branch, the release branch should only receive bug fixes from this point forward.

3.5 Release preparation

The release manager removes the "-SNAPSHOT" suffix from the release branch version number and prepares whatever release artifacts are required, making them available to the development team for internal inspection and testing. In the case of Java modules, JAR files that are ultimately destined for Maven Central when the release is approved should be uploaded to a temporary *staging repository* on Sonatype OSS but not yet released.

At this point the release manager issues a call for review. Developers inspect the candidate release artifacts and check that they are suitable for release. This checking process could include the following (non-exhaustive) list:

- The software compiles, runs, and performs as intended; all automated tests pass
- The licensing information is correct, and the authorship records are complete (the presence of licensing headers in source files should have been checked by Jenkins, but files added since the branch was created may have been missed)
- No proprietary data or incompatibly-licensed code has accidentally been included

If a developer is happy with the release, they vote "+1". If they identify any problems, they vote "-1" with an explanation of the issues they have found. If there are any "-1" votes the release would generally be suspended until the identified issues have been fixed, at which point a new candidate is produced

and the process begins again, but ultimately the decision whether or not to release rests with the release manager.

3.6 Completion of the release

Once a candidate version has been approved for release, the release manager tags the final version in the GitHub repository(ies) and attaches any binary packages to the GitHub release. There may be other specific actions required for certain modules, for example Java modules will require that the release manager *release* the staging repository on Sonatype OSS (which will promote the released artifacts to Maven Central), modules with Docker images should be released to the Docker hub, etc.

The release manager is then responsible for announcing the release via whatever channels are deemed appropriate – project mailing lists, Twitter, the OpenMinTeD website, etc. The release notes on the GitHub release will constitute the primary release announcement, and all other publicity around the release should link to the GitHub release page.

4. *Initial Release Timeline*

This is an initial snapshot of the proposed timeline for releases of the OpenMinTeD platform components and applications. The master timeline is hosted on the Redmine project management system, and will be updated as the project's needs and architecture plans are clarified, and as they further evolve. Any proposed changes to the timeline will be discussed in the technical work packages.

Each release will follow a two-month cycle of pilot → testing → integration → deployment, so for a release target date at the end of one month the release cycle starts at the beginning of the preceding month.

Start of release cycle	Release date	Month	Mile-stone	Type	Title
01/08/2016	30/09/2016	16	6.1	Component	Registry
01/08/2016	30/09/2016	16	7.3	Platform	Platform V1
01/12/2016	31/01/2017	20	6.2	Component	User and accounting services V1
01/12/2016	31/01/2017	20	9.1	Applications	1st Community Driven Applications
01/06/2017	31/07/2017	26	6.3	Component	User and accounting service V2
01/06/2017	31/07/2017	26	6.4	Component	Workflow service
01/06/2017	31/07/2017	26	7.4	Platform	Platform V2
01/06/2017	31/07/2017	26	9.2	Applications	2nd Community Driven Applications
01/06/2017	31/07/2017	26	6.5	Component	Annotation presentation service
01/10/2017	30/11/2017	30	6.6	Component	Annotation editing service and crowdsourcing plugins
01/12/2018	31/01/2018	32	7.5	Platform	Platform V3
31/01/2018	31/01/2018	32	9.3	Applications	3rd Community Driven Applications