D2.2

Workflow Guidelines

v1.0.1: 2016-08-11

Ilan Shimshoni (Technion/Haifa)

Stephen C. Phillips (IT Innovation)

In this document we describe the challenges introduced to the project by the geographically distributed nature of the project partners and our approach to ensuring an efficient development methodology based on "agile" principles. We propose the use of DevOps to support incremental software releases and ongoing requirements updates, and show, using examples from the GRAVITATE platform development, how this is being implemented.

Horizon 2020
Grant agreement number 665155

gravitate-project.eu

Project acronym	GRAVITATE
Full title	Geometric Reconstruction and Novel Semantic Reunification of Cultural Heritage Objects
Grant agreement number	665155
Funding scheme	Research and Innovation Action
Work programme topic	H2020-REFLECTIVE-7-2014
Project start date	2015-06-01
Project duration	36 months
Workpackage 2	User Requirement Analysis and Planning
Deliverable lead organisation	Technion/Haifa
Authors	Ilan Shimshoni (Technion/Haifa) Stephen C. Phillips (IT Innovation)
Reviewers	Paul Walland (IT Innovation)
Version	1.0.1
Status	Final
Dissemination level	PU: Public
Due date	PM12 (2016-05-31)
Delivery date	v1.0: 2016-07-27; v1.0.1: 2016-08-11

Version	Changes
1.0	Initial release
1.0.1	Fixed minor formatting issue

Table of Contents

1. Introduction

GRAVITATE is a complex software system developed by two research groups from the UK (IT Innovation (ITInnov) and The British Museum (BM)), a group from Italy (IMATI), a group from the University of Amsterdam (UVA), the Cyprus Institute (CyI), and group from the Technion and the University of Haifa in Israel. The project is thus geographically highly distributed. It consists of a complex architecture and on-going research efforts for which the requirements are usually not provided by the researchers themselves, who are computer scientists, but by members of other partners (BM & CyI) and external experts from the field of cultural heritage with whom the partners consult.

For such a project to succeed it has to be carefully planned taking into account the special characteristics of the project. Even though in the first year of the project requirements were collected from domain experts in the various fields of cultural heritage, additional requirements will surely turn up. It is therefore important not to assume that the standard waterfall model for software design could be used. The project will therefore use the agile project development methodology. Its main principles and how these principles apply to GRAVITATE will be described in Chapter 2. As mentioned above GRAVITATE is being developed by a geographically distributed team. As frequent face to face meetings are not possible and since a large number of developers are involved in the project, state-of-the-art development tools are being used in implementing the GRAVITATE system; tools especially designed to work well in a distributed environment. The tools that have been chosen and how they will be used will be presented in Chapter 3.

Even though the agile design methodology and the development tools that we are using are essential for a successful project, it is also important to define the interfaces between the research groups as clearly as possible and keep their number at a minimum. For these reasons the architecture of the system was designed taking that in mind. In Section 4 we will give a short review of the architecture emphasizing its distributed nature and the ability of the different partners to do most of their work without requiring them to get the consent of all the partners for each small design decision they make. Moreover, changes made in the implementation will usually not affect the other partners. This is even though the components built by the partners are closely connected, providing input to each other and working together achieve the user’s goals. The chapter will also list several types of requirements and the interactions between software components that we foresee that we will have to deal with and how the way the system is designed will enable the partners to deal with them efficiently.

2. Agile Software Development

The GRAVITATE project is committed to "agile" software development, but what does this mean in practice? The Agile Manifesto[1] is based on twelve principles:

1) The highest priority is customer satisfaction by early and continuous delivery of valuable software.
2) Changing requirements are welcomed, even late in development.
3) Working software is delivered frequently (weeks or at most a couple of months).
4) There must be close, daily cooperation between business people and developers.
5) Projects are built around motivated individuals, who should be supported and trusted.
6) Face-to-face conversation is the best form of communication within a development team.
7) Working software is the primary measure of progress.
8) It should be possible to maintain a constant pace indefinitely.
9) Agility is enhanced by continuous attention to technical excellence and good design.
10) Simplicity—the art of maximizing the amount of work not done—is essential
11) The best architectures, requirements, and designs emerge from self-organizing teams
12) Regularly, the team reflects on how to become more effective and adjusts accordingly.

In applying the Agile Manifesto to the GRAVITATE project we must address several questions:

- Why is Agile appropriate?
- Who is the "customer"?
- How do we deal with changing requirements, even late in the process?
- What technical processes do we need to deliver software frequently?
- What do we need to enable a distributed development team to work effectively together?
- How do we prioritise and ensure working software?
- How can we provide continuous attention to technical excellence and good design?

Agile development is well suited to small development teams: in the GRAVITATE project the dashboard (WP4) and platform (WP5) are led by CNR and ITInnov and the majority of these components will be developed by the two partners with UVA and BM also providing significant effort in the work-packages. Each partner has a small number of developers assigned to the project and so the communication channels are simple. The development process is confounded somewhat by the distributed nature of the development team, but on the other side we have the advantage of the developers including an end user partner (BM) as well as researchers.

Of course, the main point of Agile is to be just that: agile. In a Research and Innovation project such as GRAVITATE it is naive to think that a waterfall model can be followed where all requirements can be captured early on in the project, a design can be finalised, software created and then delivered at the end. Even an iterative waterfall model will encounter problems because of late requirements. Late requirement arise naturally from inevitable changes in research direction

[1] The Agile Manifesto: http://www.agilemanifesto.org/

when a research idea does not fully succeed or from changes in user requirements arising from users gradually understanding the potential of new technologies and techniques: we are not building a product type that is already available and understood. By building Agile processes into the development, the utility of the final software can be much increased.

The GRAVITATE Grant Agreement specified that the project will be Agile but unfortunately also encodes an iterative waterfall model into the Gantt chart. It has been agreed with the Project Officer that the requirements work-package (WP2) should be extended to cover the first two years rather than just the first year to support continued requirements capture which, even in an Agile process, is still more intense earlier on in the project. In the third year the Agile processes can be supported through the T6.4 evaluation task.

The answer to the question “who is the customer?” is complicated. One answer is that the customer is the European Commission (in that the Commission provides the funding) but we should treat the Commission more as a “sponsor” in this discussion. Agile processes are predicated on developing a product for a customer and the closest mapping to GRAVITATE is that the product is the GRAVITATE platform (incorporating the dashboard as the primary user interface) and the customer is the cultural heritage (CH) institutions both within the project (the British Museum and Cyprus Institute) and more widely.

The requirements document from the first year identifies several specific end users (customers) from CH institutions who provide requirements: the curator, illustrator, researcher and conservator. However, requirements come from less direct forms of exploitation already identified in the project involving other “customers”. For instance, exploitation opportunities of an algorithm may be increased by taking into account reassembly issues in another domain.

In the remainder of this document we describe some technical details of how to achieve an Agile but distributed development process (Section 3) and go into some of the specifics regarding how requirement changes have an impact on the different work-packages and software components (Section 4).

3. Technical Solutions for a Distributed Team

As the agile manifesto states, "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation." The very nature of collaborative European projects often enforces a distributed development environment. The added complexity introduced by this situation is often relieved through having an architecture mapping to a task breakdown in which many partners produce single components with defined interfaces which are then integrated into a "platform" by a single partner.

In GRAVITATE we are adopting this approach to some extent, for instance in task T5.3 ITInnov leads the task and will be responsible for the delivery of the functional prototype and CNR-IMATI, UVA, Haifa and Technion help with the geometric algorithm integration. Again, in T5.6, ITInnov leads the "system integration" with the assistance of the other partners.

One particular area with a complex dependency is between the user interface (a web dashboard and a high-performance desktop client) and the supporting web services. Here the partners involved (primarily CNR-IMATI and ITInnov but also Technion/HAIFA and UVA) need to work closely together to evolve the interfaces as the development process proceeds.

For communication between the developers there are four key points:

- Skype for face to face communication (all consortium members register their Skype ID in a central location).
- An understanding of each other's working hours (across time zones) and the use of the "busy" flag in Skype only when absolutely necessary with the willingness to accept conversation requests most of the time.
- The use of the Mattermost software[2] for text chat, providing persisted chat rooms linked to the version control system deployed at ITInnov.
- Clear communication of issues and software changes using the version control system.

Many of the other commitments made in the Agile Manifesto are facilitated by what is known as "DevOps" which brings development techniques and operational deployments closer together.

3.1. DevOps

An area of software engineering known as "DevOps" has come to the fore in the last few years as agile methods have gained in popularity and tools for managing virtual machines and containers have rapidly developed.

In many IT environments there are entirely different teams assigned to software development and operational deployment of software, each taking care of different concerns. Deploying a new piece of software, or a new software release, operationally is traditionally a time-consuming process involving extensive quality assurance. DevOps encourages the use of the same tools and processes in the development phase as in the QA and operational deployment phases, thus smoothing the

[2] Mattermost: http://www.mattermost.org/

journey from one to the other. This has been driven by the demand from agile processes to make many small incremental releases.

In GRAVITATE, the software being developed in WP4 and WP5 will be deployed in WP6. The key stages of development and deployment and the associated tools to be used are as follows:

- Code:
    - Version control through GitLab[3] hosted by IT Innovation
    - Developer communication via Mattermost[4], hosted by IT Innovation
    - Code review in GitLab
    - Continuous integration on master branches through GitLab
- Build
    - Gradle build tool automating the build and compile-time dependencies
    - A common environment will be provided by Docker[5] containers and/or Vagrant[6] virtual machines with run-time dependencies managed with Ansible[7]
- Test
    - Unit tests and system tests with Junit
    - Further code quality checks will be performed using SonarQube[8], deployed at IT Innovation
- Package
    - Gradle will also build documentation and package the software
- Release
    - Will be done manually
- Deploy
    - To be done with Docker and/or Vagrant

GitLab provides an excellent interface to the Git version control system as well as many other useful features. Git is very flexible and does not impose any particular branching style but certain ways of working must be defined for a team.

There is a very good description of some of the common Git branching workflows on the Atlassian site[9] which also documents the Git command line commands required to perform the various

[3] GitLab: https://about.gitlab.com/
[4] Mattermost: https://www.mattermost.org/
[5] Docker: https://www.docker.com/
[6] Vagrant: https://www.vagrantup.com/
[7] Ansible: https://www.ansible.com/
[8] SonarQube: http://www.sonarqube.org/
[9] Git worlkflows: https://www.atlassian.com/git/workflows

operations. The "Feature Branch Workflow" is a fundamental pattern which we will adopt. In this pattern when a feature is to be added then a branch is made from the master (with a name describing the feature) and the development work goes on the branch. Generally just one developer will work on a feature but this is a preference to avoid conflicts, not a constraint. If it seems that more than one developer should be working on a feature then we should consider if the task can be logically split into smaller features. Once the main developer of a feature thinks it is ready to merge back into the master a "pull request" (sometimes called "merge request") is issued. Other developers then do QA on the feature's code and if it passes then the merge is made and the master is updated.

In Figure 1 below a branch called "feature/annotate" is created, worked on and merged into the master. Ideally, feature branches are small, independent and merged quickly but sometimes a feature (such as "feature/search" in the Figure) takes a long time. In these cases it is acceptable to pull from the master into the feature branch to bring in new code that the feature depends upon.

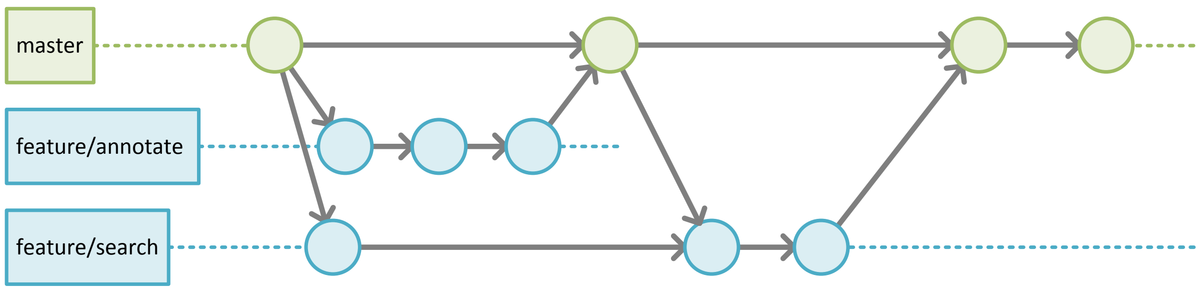

Figure 1. Git feature branch workflow.

The well-known "Gitflow" workflow builds on the Feature Branch Workflow to add in a "Develop" branch for merging features into, a "Release" branch for working on releases (not features) and uses the "Master" branch for storing the tagged releases themselves. It also defines "Hotfix" branches, branched off the Master for fixing issues in releases. Here we propose a variation on the Gitflow workflow which allows for the maintenance of multiple releases.

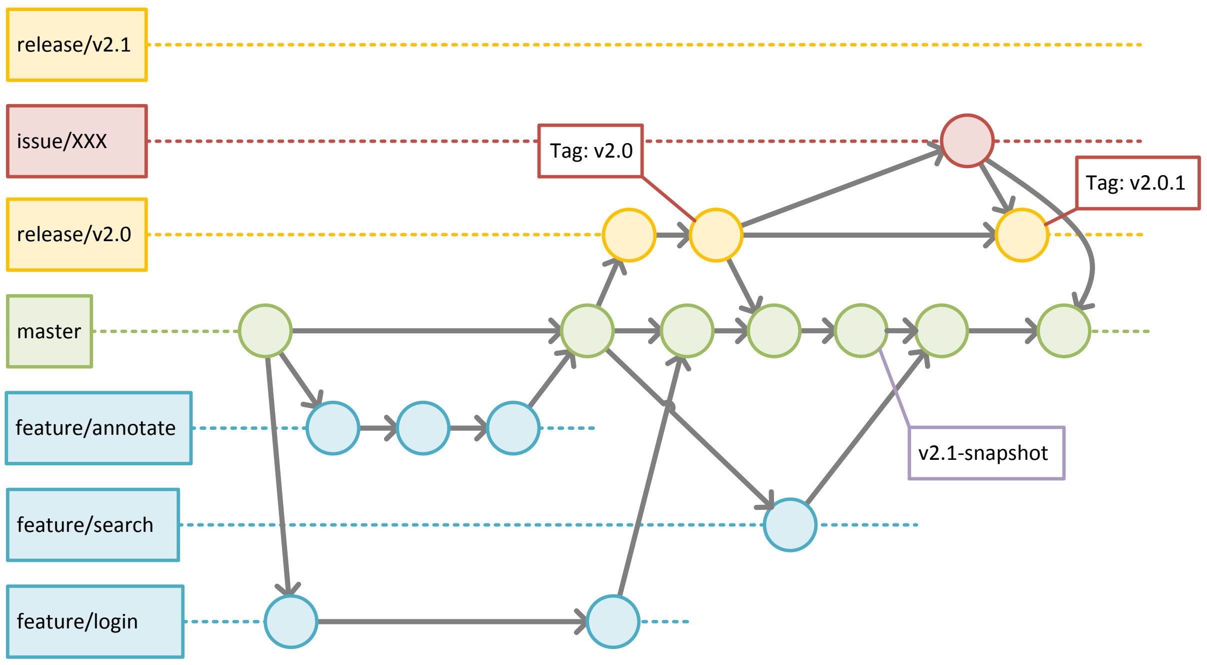

Figure 2. Proposed Git workflow.

4. Workflows for Requirements

Due to the research nature of the GRAVITATE system, its architecture was designed in a loosely coupled manner. This way, modifications in various components which are due to direct user requirements, or new datasets or use cases which yield changes in requirements can be performed independently by the partners without requiring system-wide integration. There are of course cases in which system-wide modifications will be needed but those should be as rare as possible.

The GRAVITATE system consists of the following components.

- The Graphical User Interface (GUI) which is being designed and implemented within WP4 and WP5 by ITInnov and IMATI.
- The ResearchSpace database and query tool which is being modified to address the special needs of GRAVITATE. This component is being designed and implemented within WP3, WP4, and WP5 by BM IMATI and ITInnov.
- The underlying GRAVITATE infrastructure. This is designed and implemented by ITInnov within WP5.
- The data repository. This includes the triple store database, the scans of the artifacts and their images, and stored computed features and their indices. This repository is managed by the GRAVITATE infrastructure. This repository is populated mainly by the cultural heritage partners CyI and BM, while the features are designed by IMATI.
- The algorithms. In order to make the project manageable from the software engineering point of view, most of the algorithms developed within WP3 and WP4 are invoked as executables by the GRAVITATE infrastructure. The algorithms will be developed by the IMATI, UVA, and Technion partners.

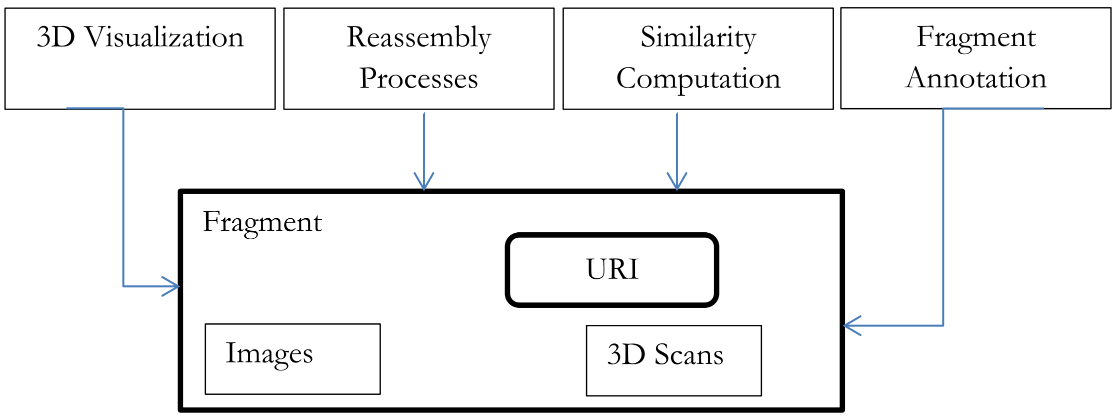

Figure 3. The interface between the fragment software library and GRAVITATE components

Each of these components will be managed by its owner as a separate project using the software engineering tools described above. In order for the project as a whole to be manageable the interfaces between the components have to be well defined on the one hand and flexible on the other hand.

In the following sections we give some important examples on how this can be achieved:

1) We will first consider the interface between the data repository and the other GRAVITATE components such as the GUI and the various algorithms, illustrated in Figure 3. When a GRAVITATE user will want to add a new fragment to the repository, it will include besides the textual description (metadata) 3D scans and accompanying images. These will be acquired according to GRAVITATE guidelines, but the user will have the freedom to make decisions on how this is done depending on the 3D and 2D features that are required to be visible. According to those decisions the scans and images will be generated. Scans might also be maintained in several resolutions generated from the original scan. Since these scans and images will be used by most of the components of GRAVITATE, where each of them might have different requirements on the quality of these objects, it is important for each application to be able choose the most suitable scan or set of images. This requires that the object representing the fragment (part of the GRAVITATE infrastructure) will be able to communicate information related to the scans and images to the application so it can automatically choose the most appropriate one; for example, for visualization low resolution scans are needed. Another algorithm might want higher resolution scans but not too complex since its running time depends on the size of the mesh. Thus, when given the URI of a fragment, the algorithm will be able to choose from the available scans and images and their attributes which ones to use.
2) One of the basic user requirements for the GRAVITATE system is that given a set of objects, the GRAVITATE system will cluster them in an archeologically meaningful manner within the reassociation process. This vague requirement has to be spelled out by examples and use cases. Quite a few of these examples can be generated using the Salamis data set but other data sets might also be needed. The clustering will be performed using both the metadata and similarity measures based on combinations of similarity measures of geometric features computed from the scans and the images. It is therefore quite possible that in order for the system to successfully cluster the fragments according to the user's requirements new geometric and colour features and similarity measures might be needed. It is therefore important that when it is decided to add or remove such a feature or similarity measure to the system, a standard procedure will be available to perform it. This means that the procedure (which will be given as input to the function which computes the feature) will compute the new measure for all existing fragments, save the results in the repository and for all new fragments this feature will also be computed. Once this new feature or similarity measure has been added, the applications that use it to perform similarity should be able to use it without modification. Here again a flexible interface between the computed features stored in the repository and the algorithms will be provided.
3) Considering the two examples above, they are examples of the interface between the data repository and the other components of GRAVITATE. Thus, an input/output library with these characteristics will be implemented and used by the algorithms to access the repository. Thus, if it will be decided to make a modification to the structure of the repository, only this library will have to be modified leaving the all the other components that use it unchanged.

4) Another source of changes to the system that can be required by GRAVITATE users or by the GRAVITATE partners are at the high and middle levels of the system. Consider the following two examples from the realm of reassembly, although the idea is general: Examples of changes at the high level could include trying several variants of the reassembly module and user interface. In the final version of the system only one or two of them will remain, but it could be very useful to let users experiment with more variants so an intelligent choice can be made. It is also possible that different types of users might be interested in different variants of the algorithm or its user interface. At the middle level, the premating component could try to work with several variants of the mating algorithm. This should be done to make experimentation possible. The final decision on what will go into the GRAVITATE system will of course be made by the partners based on the performance of the components and how well they satisfy the user requirements.
5) The collections on which the GRAVITATE system operates come from several different museums, where each one has its own database with its own schema. One of the main strengths of the ResearchSpace software package is its ability to work with different database schemas in parallel. Thus, a query given by the user to ResearchSpace is translated into appropriate queries which can be answered by the different databases. We have already started working on incorporating collections from CyI, BM, the Ashmolean Museum, and the Fitzwilliam museum under GRAVITATE. If it will be required to add more collections to GRAVITATE, all the work will be done by the BM partner who is the ResearchSpace expert with some help provided by ITInnov.
6) The user might make requests to modify the system's user interface. If the requirement is limited only to the user interface, it can be done within work packages 4 & 5 by IMATI and ITInnov. If however, as it sometimes happens, the modification will require extended algorithmic capabilities, the relevant partner will be asked to provide it as part of its work in WP3.

This list of possible requirements and modification requests is not complete and more examples may be considered, but we believe that since we will be able to deal with the examples given above without too much disruption to the other project members working on other components of the system, we will likewise be able to deal with other change requests in a similar manner.

5. Conclusions

In this document we described the geographically distributed nature of the GRAVITATE project members and the problems it brings in developing a state-of-the-art software tool which includes a novel architecture and whose components are currently still in the research phase. Moreover, the system requirements are not given by the developers themselves, but by the other partners and external cultural heritage experts. In order for such a project to be a success these challenges have to be addressed from the beginning of the project. The project is therefore being developed using the agile methodology which is designed to deal with new requirements elicited at various times during the development of the system and not all collected at the beginning of the development of the project. We are using state-of-the-art software development tools which enable groups of developers to work together and to discuss and document requirements for changes in the system. The architecture of the system was designed taking into account the distributed nature of the development team, enabling its members to work independently most of the time while still enabling them to cooperate closely in solving their research questions and provide a cohesive system which addresses the users' requirements and meets their expectations. In the document we gave a list of possible requirements and requests for change that we expect that we will have to address and how the project members will be able to address them efficiently. There is of course no way to know that we have considered all the types of modifications that we will have to deal with, but we believe that we will be able to deal with them without too much disruption if and when they occur.