Mastering Data-Intensive Collaboration and Decision Making

FP7 - Information and Communication Technologies

Grant Agreement no: 257184
Collaborative Project
Project start: 1 September 2010, Duration: 36 months

D5.1.1 - Standards and guidelines for development (initial version)

Due date of deliverable:	30 April 2011
Actual submission date:	29 April 2011
Lead Partner:	NEO
Contributing Partners:	CTI, FHG, NEO, UPM

Nature: ☒ Report ☐ Prototype ☐ Demonstrator ☐ Other

Dissemination Level:

- ☒ PU : Public
- ☐ PP : Restricted to other programme participants (including the Commission Services)
- ☐ RE : Restricted to a group specified by the consortium (including the Commission Services)
- ☐ CO : Confidential, only for members of the consortium (including the Commission Services)

Keyword List: Software quality, Coding guidelines, Test-driven Development (TDD), Test coverage, Scrum, Agile methodologies, Software configuration management, Continuous integration, Build management, Web services, SOA, REST, SOAP, Issue tracking, Integration, Collaboration tools, Maven, Subversion, Hudson, JIRA, Confluence, FindBugs

The Dicode project (dicode-project.eu) is funded by the European Commission, Information Society and Media Directorate General, under the FP7 Cooperation programme (ICT/SO 4.3: Intelligent Information Management).

The Dicode Consortium

Research Academic Computer Technology Institute (CTI) (coordinator), Greece

University of Leeds (UOL), UK

Fraunhofer-Gesellschaft zur Foerderung der angewandten Forschung e.V. (FHG), Germany

Universidad Politecnica de Madrid (UPM), Spain

Neofonie GmbH (NEO), Germany

Image Analysis Limited (IMA), UK

Biomedical Research Foundation, Academy of Athens (BRF), Greece

Publicis Frankfurt Zweigniederlassung der PWW GmbH (PUB), Germany

Document history			
Version	Date	Status	Modifications made by
1	13-04-2011	First draft	Guillermo de la Calle Velasco (UPM) Isabel Drost (NEO) Axel Poigné (FHG) Doris Maassen (NEO) Manolis Tzagarakis (CTI)
2	14-04-2011	Second draft (sent to internal reviewers)	Doris Maassen (NEO)
3	21-04-2011	Reviewers' comments incorporated (sent to SC)	Doris Maassen (NEO)
4	28-04-2011	SC's comments incorporated	Doris Maassen (NEO)
5	29-04-2011	Final (approved by SC, sent to the Project Officer)	Doris Maassen (NEO)

Deliverable manager

- Doris Maassen, NEO

List of Contributors

- Guillermo de la Calle Velasco, UPM
- Isabel Drost, NEO
- Axel Poigné, FHG
- Doris Maassen, NEO
- Manolis Tzagarakis, CTI
- Spyros Christodoulou, CTI

List of Evaluators

- Duncan Russell, IMA
- Anastasia Kastania, BRF

Summary

This deliverable defines the standards and guidelines for software development agreed on by all Dicode partners. Besides the current members of the Dicode development teams the deliverable also serves the needs of both new staff joining the project and developers from outside the consortium who contribute to the open source libraries developed in Dicode. The topics discussed range from coding conventions over test-driven development and tool support to Agile methodologies like Scrum and Kanban. This deliverable is designed as a "collaboratively living document" that will be continuously updated by all technical partners during the project.

Table of Contents

1 Guidelines to development – overkill or fundamental necessity?

The establishment of some guidelines to develop software constitutes a fundamental necessity in collaborative projects with several partners involved in development tasks. Maintenance and retraining represent the major costs of software development. Usually, software is not maintained by the original author but different persons. Developing applications or writing code without following a set of essential agreements may lead to a complete chaos. The appropriate definition of such agreements or guidelines (and its subsequent implementation) might be the difference between the success and the failure of a project.

These guidelines are intended for technical partners and all staff involved in the development of software products within the project. They constitute the main reference for analysts, system designers and technical developers, both for those involved in the project from the beginning and for new staff joining the project. Every time new members join the development team, they have to catch up with the work already carried out. Development guidelines are especially useful in those cases. Among others, the main benefits achieved with the definition of this kind of guidelines are:

- Ensuring software quality by defining a common and standard development methodology;
- Improving the legibility and understanding of the code;
- Improving the productivity of the programming staff, and
- Reducing the time dedicated to detecting and correct errors, i.e. reducing costs.

Therefore, agreements have to be achieved at different levels, ranging from the programming language to be used to implement the system or the software versioning system used to the application servers more suitable to fulfil the necessities of the project or even the physical place where these application servers will be deployed.

Dicode is a collaborative project where the technical partners will develop an integrated system with the support, supervision and collaboration of the rest of the partners. As mentioned above, it is crucial to establish a set of rules to be followed by all partners before the start of software development. Main programming language, integrated development environment (IDE) and software versioning system to be used, application servers, developing frameworks and so on, are some of the recommendations that can be found in the current document. Due to the intrinsic nature of Dicode, free and open source resources, tools and frameworks will be specially considered to be adopted within the project. Additionally, an agile and collaborative development methodology for improving the outcomes of the project regarding the software products will be described.

In Dicode, we envisage the current deliverable as a "collaboratively living document" that will be continuously updated by all technical partners during the project. (An enhanced version of this deliverable, namely D5.1.2, is due in month 30 of the project). The document is divided into several sections covering different issues related to software design and implementation. These sections deal with topics such as software quality, development principles, agile software development, tools and integration issues. At the beginning, each section has been completed by the technical partner whose expertise or skills match better to

the topic of the section. To facilitate the distribution, accessibility and updating of these guidelines, the current version will be maintained at the Dicode's wiki, available at: https://wiki.dicode-project.eu/display/DIC/D5.1+-+Standards+and+guidelines+for+development.[1]

In the following sections we present the initial version of the Dicode's development guidelines and standards agreed on by the whole consortium to develop and implement the foreseen architecture and system.

2 Software quality

From a customer's perspective, it seems easy to distinguish high quality software from low quality products: First and foremost, a software product must conform to requirements it was designed to meet. Software pieces that do not solve customer's problems are useless. When dealing with data analysis setups, it is also necessary to count in scalability to the anticipated data set sizes. On the other hand, when used by multiple users as it is common in a web application setting, the developed solution must scale to the predicted number of users interacting with the system.

Of course, it is needless to say that the product must work correctly, it must be complete and it should be free of bugs. A set of software metrics helps to define and permanently monitor this requirement. In case of faulty user input or hostile environments, the developed software must be fault-tolerant and reliable. The release 1.0 of any software project is just the very beginning of a product's life cycle. As a result, it must be designed for future extensibility, maintainability and must be well documented.

When looking at a piece of code from a software developer's perspective, a few additional requirements come to mind. The code must be understandable and clear. This requirement is one of the basic preconditions of making software extendible and maintainable. The implementation should be concise, that is avoiding any code or configuration duplication. Notation should be consistent across the whole implementation. This again makes code easier to read, makes it easier to change the system at a future point in time and leads to less complex software. To reach the goal of minimizing bugs in a system, it greatly helps to write code that is easy to test automatically from the start.

When looking at today's hardware and systems market, it quickly becomes clear that software products must be portable, i.e. easy to run on different hardware and software configurations. When put out in the wild, the system should work efficiently, not introduce any security holes that might lead to leaking internal data or abuse.

The goal of Dicode is to develop software that runs on a large scale and can be re-used even after the project has long finished. As a result, ensuring high quality of any software produced is of large importance to the project. This document aims to set the standards and guidelines for development at Dicode that ensure high software quality throughout the project.

[1] This document has been exported from the Dicode wiki. As we are striving for interoperability between Wiki and Office, external web pages (e.g. those of software development tools) will be referenced by hyperlinks and not by footnotes. This allows for a seamless re-import of the reviewed deliverable into the Dicode wiki.

2.1 Coding conventions

In accordance with the goal of developing code that is concise and consistent, we are planning to adhere to common coding conventions. For the sake of simplicity and consistency with other Java projects, we adopt the Sun Java Guidelines. The only change we made for the sake of conciseness is to use two spaces instead of tabs for indentation – a convention heavily used by open source projects, e.g. Apache projects. Tools such as Checkstyle can be used to ensure adherence to the coding conventions. The goal is to keep Checkstyle warnings in each Java file to zero. For source code developed in programming languages other than Java, standard coding conventions are being used where appropriate. Having a common style across all code files makes it easier for developers to read other developers' code as the same formatting patterns can be found in the source code.

Coding conventions themselves are primarily focused on code formatting. They are inherently fine grained and detailed. To ensure standardization on a broader level, the Dicode members agree to use common design best practices, that is design patterns. Those patterns describe commonly occurring problems in software development that can be solved by common design choices. The advantage of having named design patterns is the decreased complexity when trying to understand existing code. It also makes it easy to communicate one's design choices to other developers.

One last principle we are trying to implement is to keep it simple and avoid overly complex designs. It has been shown in the past that it is impossible to anticipate all possible use cases and configuration needs of any piece of software. As a result, the idea of keeping components simple and separate is to make re-design and re-factoring a lot easier and thus be prepared for changing requirements and system environments.

2.2 Development principles

In this section, the main development principles of Dicode are described: Test-driven development and the adherence to a strict "Definition of done" concerning the completion of user stories as proposed by the agile software methodology Scrum.

2.2.1 Test-driven development

We strive for 100% test coverage. Coverage analysis will be performed based on Clover, Cobertura, or similar tools. Both tools come with IDE and Maven integration. Being established as an open source project, Dicode is granted to a cost-free license of Clover from Atlassian.

2.2.2 Definition of Done

A user story is defined to be done if it has test coverage of at least 90%, has no more than zero FindBugs or PMD warnings and has no more than zero Checkstyle warnings. If the user story deals with contributing code back to an external project, it is considered done as soon as the patch has been accepted and integrated upstream.

2.3 Tool support

Tools usage includes established local standards. The following tools were chosen by Dicode:

- Atlassian JIRA for project organization and as an issue tracking tool (Required).

- FindBugs statically analyses code for potential bugs. Integration with both Eclipse and IntelliJ IDEA is available. Publishing on a Maven generated site is possible through the respective site plug-in. Publishing on Hudson reporting is possible through a plug-in as well. The goal is to keep FindBugs issues to a minimum, zero where ever possible. The configuration to use is checked into the Dicode code base (Recommended).
- PMD is a second tool for static code analysis that comes with support for Maven and all major IDEs. Bugs should be kept to zero here as well (Recommended).

3 Development process standards

Dicode strives for regular integration. On the work package level, we are having iterations of three weeks. At the end of each iteration, we must have an integrated, running and deployed version of our software.

3.1 Agile software development

Agile development emphasizes on working software/system as the principal measure of progress, on software being delivered frequently with the focus on collaboration with the customer, and on responding fast to changing requirements resulting from experimenting with the available software. In Scrum development is essentially cyclical with rather short feedback loops as Figure 3.1 illustrates.

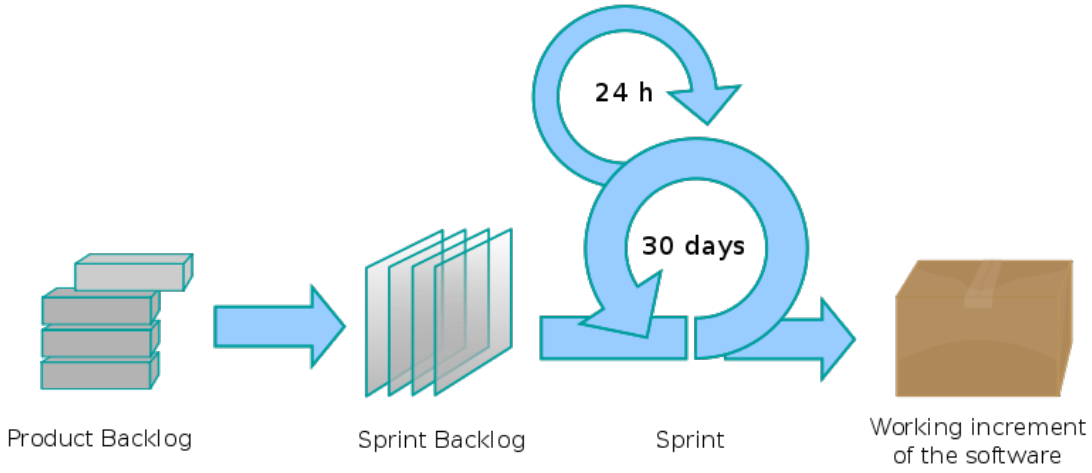

Figure 3.1 The Scrum process

During each "Sprint" (the length of which is decided by the team), a potentially shippable product increment is created. The features that go into a Sprint come from the product "backlog", which is a prioritized set of items of work to be done. Which backlog items go into the Sprint is determined during a Sprint planning meeting.

Further development principles from the agile world include:

- User stories are not estimated in terms of amount of time needed but in terms of complexity. The number of complexity points that one may give are computed according to Fibonacci series - it starts small but gets bigger faster the larger the

numbers get. This takes the human incapability of accurately estimating large items into account.
- Priorities of user stories are assigned not by the team but by the users according to user's business value. As all user stories are ordered by priority, the above estimation is easy: only the most important ones have to be estimated to get the most business value out of each iteration.
- Tracking of user stories done (and accepted as done by the users) leads to a velocity graph that enables teams to estimate the amount of user stories that can be done in the next iteration based on each story's complexity.
- A retrospective at the end of each iteration uncovers problems and helps optimizing the development process for faster and better delivery.
- The risk of user stories not being accepted can be minimized by integrating the product during iteration and doing incremental releases.
- The users are responsible for defining what to implement, not how to implement it. That means: No technological or tooling requirements are pre-defined by those who are not actually developing software in the respective work package.

Scrum focuses on allocation of work items in a time frame but provides no mechanism for mapping work items to the available workforce. For the latter, Dicode will complement Scrum by the Kanban methodology[2], the principles of which are:

- Visualize the workflow.
  - Split the work into pieces;
  - Use named columns to illustrate where each item is in the workflow.
- Limit Work In Progress (WIP) - assign explicit limits to how many items may be in progress at each workflow state (Sprint)[3].
- Measure the lead time (average time to complete one item, "cycle time" in Scrum), optimize the process to make lead time as small and predictable as possible.

The combined methodology adds flexibility in that a strict time management organized in Sprints may be optional: it can have separate cadences for planning, release, and process improvement. It can even be event-driven instead of time-boxed and items may be added in an on-going iteration. On the other hand, Scrum introduces a more rigid time discipline as compared to Kanban and prioritization of backlogs is required.

3.2 Open development - embedding Dicode in the open source community

Dicode as a project is not only re-using quite a large amount of free software. Its goal is to publish any software developed under an open source license. However, simply putting a project under an OSS license and publishing it on the web is no use when striving for sustainability and future usability.

With the help of the organization OSS Watch, an advisory service for open source projects, the consortium attempts to embed Dicode in the open source community. OSS Watch describes itself as an "unbiased advice and guidance on the use, development, and licensing of free and open source software". OSS Watch provides its services free-of-charge to UK higher and further education, but has a large collection of guidelines available to the general public. The following documents are especially of interest for the Dicode project:

[2] For a discussion about combining both methods see http://www.slideshare.net/techdude/kanban-vsscrum.
[3] See http://www.slideshare.net/agilepartner/agile-mtteg-series-session-4 for a more detailed description.

- Essential tools for running a community-led project ;
- Avoiding abandon-ware: getting to grips with the open development method ;
- Open source and research infrastructure ;
- Sustainability lessons for research infrastructure ;
- Community lessons for research infrastructure.

4 Technical development environment

4.1 Tools supporting automation

4.1.1 Build system

Apache Maven is a tool to support the process of software development. In particular, it is a software project management and comprehension tool aiming to manage the build processes, documentation, reporting, dependencies, software versioning system, releases and distribution. The objectives of Maven are to make the build process easy, provide a uniform build system, supply quality information on projects and allow transparent migration to new features. Extensive documentation is available [1].

Even though Maven was in first place developed to be used for building Java projects, it may also be used to support projects written in a number of other languages such as C#, Ruby and Scala. Support for these languages comes in the form of specialized plug-ins. The Maven project is hosted by the Apache Software Foundation and has been part of the Jakarta project. Maven provides a wide range of plug-ins, which can affect the Maven Lifecycle, offer access to goals, and perform tasks such as source compiling, packaging and publishing to sites.

The conceptual foundation of Maven is the Project Object Model (POM), which provides the necessary abstractions to represent and capture all project related aspects. The POM specifies what sort of project is targeted and how to modify the default behaviour to generate output from the sources. In Maven, XML is used to instantiate the POM. Such XML files contain information for each project, such as where the sources are located, how to build the project, what the target should be, where the test source directory is located etc. While the POM is basically used for Java projects (as Maven's default plug-ins aim at building JAR files from source files), it can be also used for building, for example, C# sources. POM XML files are structured and contain four categories of description and configuration. These include:

- General project information: the project's name, URL, the sponsoring organization, the list of developers/contributors and the license of the project ;
- Build settings: the behaviour of the default Maven build, the location of source and tests, information concerning the addition of new plug-ins, plug-in goals, site generation parameters;
- Build environment: profiles that can be activated for use in different environments, build settings for specific environments;
- POM relationships: dependencies of the current project on other projects, POM settings inherited from parent projects, submodules.

The concept of "build lifecycle" is central for Maven. In particular, there is a clearly defined sequence of phases/steps that give order to a set of goals, which for example include building and distributing a particular project. A "builder" of the project has to learn a small set of commands to build the Maven project, as long as he has created the necessary POM file

which will ensure the appropriate parameters of the building process. There are three built-in build lifecycles in Maven: default (project deployment), clean (project cleaning) and site (creation of project's site documentation). The default lifecycle has 8 phases (validate, compile, test, package, integration-test, verify, install and deploy), the clean 3 (pre-clean, clean and post-clean) and the site lifecycle 4 (pre-site, site, post-site and site-deploy).

A build phase reflects to a specific step in the build lifecycle. To specify the way a build phase performs these tasks, goals have to be defined. A goal is a specific task contributing to the building/managing of the project and may be related to zero or more build phases. A build phase may have zero or more goals bounded to it. There are two ways to assign tasks (goals) to build phases: packaging (for example jar, which is the default, war, ear and pom) and plug-ins.

Several plug-ins have been developed to provide integration with popular IDEs such as Eclipse, Microsoft Visual Studio, NetBeans, IntelliJ IDEA and JBuilder. Some of them provide POM file editing and POM using to determine the project dependencies within the IDE.

In the context of Dicode, Maven is in particular an interesting solution, as it can address the project's automation and configuration needs. The source code in Dicode will be developed in different languages, that include Java and C#; under such circumstances, Maven can help immensely in setting up a uniform and transparent build system.

4.1.2 Continuous Integration

Hudson is an open source "continuous integration" (CI) server that monitors executions of repeated jobs, including building of a software project or jobs run by the cron demon. Recent Hudson releases focus on two jobs:

- Building/testing software projects continuously by providing a continuous integration system (developers integrate changes to the project and users are able to get a fresh build immediately);
- Monitoring executions of externally-run jobs (such as cron and procmail jobs).

Hudson provides support for Source Control Management tools such as CVS, Subversion, Git and Clearcase. It can execute shell scripts, windows batch commands as well as Apache Ant and Apache Maven projects. By default, 4 plug-ins are installed on Hudson, supporting CVS, Subversion, Maven and SSH. A plethora of plug-ins (more than 200, which include Artifact Uploaders, Authentication, Build Notifiers, Build Reports, Build Tools, Build Triggers, Build Wrappers, Cluster Management etc.) have been developed to extend Hudson core functionality, including plug-ins for the most popular version control systems, bug databases and build tools.

Concerning Dicode, Hudson can be used to build the developed software, run unit tests and code analysis on each check-in. The tools supports also tracking build history, which is critical for the project. For a detailed introduction, see [2].

4.2 Tools supporting project communication

4.2.1 SCM System

Software Configuration Management (SCM) refers to the task of tracking and controlling changes in software. The main goals of SCM include establishing a controlled change process, build and process management, tracking bugs and defect as well as coordinate

interactions of the development team related to development processes. The most commonly configuration management practices include revision control and the establishment of baselines i.e. significant states in the revision history of the items developed and configured.

With respect to Dicode, a Subversion-based code repository will be deployed to control changes in the project's source code. In order to address the project's open source requirement, a publicly accessible repository has already been instantiated. In particular, a Google code repository has been setup to support the project's SCM requirements; it can be accessed at http://code.google.com/p/dicode/.

4.2.2 Internal communication

A dedicated wiki (https://wiki.dicode-project.eu) based on the Confluence platform (provided by Atlassian) has been set up for documentation of the project artifacts and for supporting internal communication among the distinct teams which form the Dicode consortium. The wiki provides a number of useful features among which space and page creation, categorization, tagging, access control, content uploading, a rich text editor for text content creation, grouping of content items, a number of macros to support extra functionality, notifications upon content creation/editing and a searching mechanism based on user defined criteria.

A number of mailing lists have been set up to provide communication among the partners participating in each of the project's work packages. There is also a global Dicode mailing list for all members of the Dicode project. Mailing lists are used for informational purposes and support discussions among members

4.2.3 Issue Tracker

The Dicode project web portal also provides an entry point to an issue tracking system, to be used for various project management purposes (such as software development tracking, repository and tracking of project's documents, etc.). The overall aim of issue tracking is to manage and maintain work units which aim at improving a system.

The JIRA tool, the issue and project tracking tool selected to be used in Dicode and which includes the GreenHopper plug-in, is provided by Atlassian and supports tracking of different kinds of issues. An issue may represent a project task, a helpdesk ticket, a leave request form etc. Each issue has a status indicating where the issue currently is in its lifecycle. An issue starts as "open" and progresses to "in progress", "resolved", "reopened" or "closed". Each issue can be resolved in many ways. The default resolutions include "Fixed", "Won't Fix", "Duplicate", "Incomplete" and "Cannot Reproduce". A JIRA project is a collection of issues. A number of distinct projects (software development project, marketing campaign, helpdesk system, leave request, management system, and web site enhancement request system) exist, depending on the user requirements.

Within Dicode, each work package will create its own issue tracking project for each artifact being developed. Releases in JIRA will be kept in sync with WPs development iterations, which in turn will be kept in sync with official software releases.

5 Integration guidelines

One of the firsts tasks carried out in the Dicode project was to survey the state of the art of the technologies dealing with data intensiveness in collaboration and decision making settings. The results of this survey were compiled in deliverable D2.1. One section of this

deliverable was dedicated to integration issues. There, the different integration approaches regarding data integration and software integration were analyzed. According to such analysis and considering the nature of Dicode, it was decided to follow a distributed approach based on a Service Oriented Architecture (SOA) to achieve the goals proposed in the project.

Currently, there are two main trends to develop SOA-based systems: WS-* (also named SOAP) and REST/RESTful services [3]. Both offer different benefits and drawbacks. Mainly, both WS-* and REST are based on well-established and proven Internet standards. WS-* presents a more complete and "heavy" definition of the services and REST uses a more lightweight style. Both approaches are based on (web) services as shown in Figure 5.1.

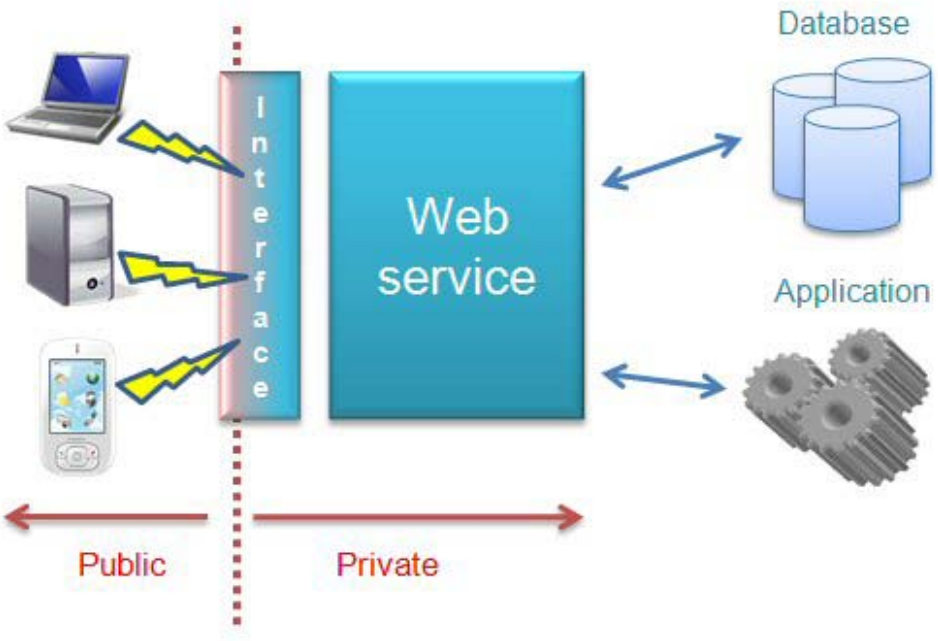

Figure 5.1 Web services (general schema)

A web service can be seen as a "Black box" that provides a concrete functionality. It defines a public interface accessible by any client software via a communication network. Such client software knows how to invoke the service and the results returned but it doesn't mind how those results are obtained. Behind the interface, the web service can be implemented using any technology, platform or language. Its functionality can be as simple as performing the addition of two numbers or as complex as accessing multiple distributed databases or executing complex data mining algorithms. This feature enables web service to develop systems loosely coupled, platform independent and highly interoperable and reusable. Web services have the lifecycle shown in Figure 5.2.

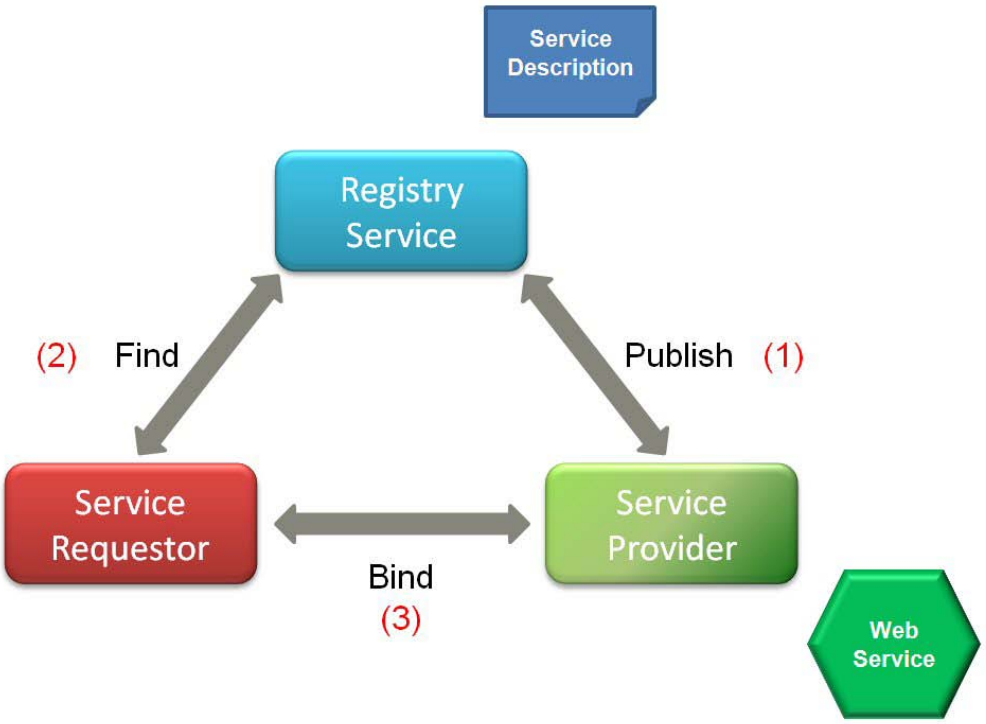

Figure 5.2 Web service lifecycle

The scenario is always the same: first, a service provider develops a new service and deploys it on a server. Then, the service provider publishes the service description to a well-known registry to allow clients to locate it. After that, service requestors ask the registry to locate services to fulfil their necessities. The registry returns information about those services. And finally, the service requestor uses the service. There are a big number of services around the world following these two specifications and most of them are publicly available through the Internet. In Dicode, we would like to take advantage of the possibility of reusing already existing resources, services and tools. For that reason, we propose an open architecture suitable for integrating both REST and WS-* technologies.

6 Summary and Perspectives

This document describes a set of practices and tools which will be used by all Dicode partners involved in software development. As stated before, we envisage the current deliverable as a "collaboratively living document" that will be continuously updated by all technical partners during the project.

Besides the current members of the Dicode development teams, this deliverable also serves the needs of both new staff joining the project and developers from outside the consortium who contribute to the open source libraries developed in Dicode. Dicode itself makes heavy use of various open source tools, libraries and frameworks. In most cases, those projects offer extensive documentation on the web. Therefore, we decided to keep the deliverable rather short and refer to publicly available external sources when possible.

Continuous updating of this deliverable is essential for two reasons: first of all, the demand for standardization and guidelines will grow with the advancement and complexity of the

project. Second, the technologies described are subject to change and new tools will enter the market which will be widely adopted in the software industry and will shape the future of agile development in the open source field.

Dicode aims at an improved integration of development tools. A tool bases workflow should force developers to follow the commonly defined process and allow for complete traceability of changes. Therefore we suggest the following steps for the next iteration of this document:

- Visualization: The current development process workflow and the tools used at each stage should be visualized. This visualization should demonstrate the actions required to close each stage.
- Development of a common metrics of continuous code assessment: Suggested metrics are
    - JIRA: Open issues, closed issues, reworking;
    - Subversion: Changes, lines of code, etc.;
    - Test coverage: Number of test cases, number of classes covered, not covered, code coverage;
    - Checkstyle warnings.

Dicode's developers strive for a seamless integration of all tools. A more detailed strategy of tool-based collaboration has to be developed in the following months.

References

[1] Van Zyl, J. (2008-10-01), Maven: Definitive Guide (first ed.), O'Reilly Media, pp. 468, ISBN 0596517335, Retrieved from http://www.sonatype.com/books/maven-book/

[2] Smart, J. F. (2010) Jenkins: The Definitive Guide, Retrieved from http://www.wakaleo.com/public_resources/continuous-integration-with-hudson.pdf

[3] Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, CA. Retrieved from http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm