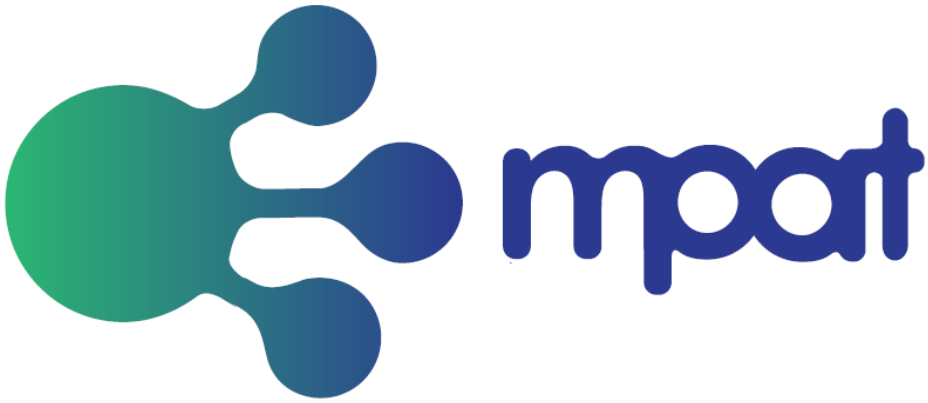

Developer tools and development guidelines

Deliverable D4.5

MPAT File ID:	MPAT_D4.5_Developer-tools-and-development-guidelines.docx
Version:	1.0
Deliverable number:	D4.5
Authors:	Stefano Miccoli (Fincons)
Contributors:	Matthew Broadbent (ULANC), Jamie Jellicoe (ULANC), Fabian Schiller (IRT), Miggi Zwicklbauer (Fraunhofer)
Internal reviewers:	Matthew Broadbent (ULANC), Jamie Jellicoe (ULANC), Fabian Schiller (IRT), Miggi Zwicklbauer (Fraunhofer)

Work Package:	WP4
Task:	T4.3
Nature:	R – Report
Dissemination:	PU – Public
Status:	Final
Delivery date:	31.03.2016

Version and controls:

Version	Date	Reason for change	Editor
0	04/03/2016	First release	Stefano Miccoli
1	11/03/2016	Lancaster university contributions	Matthew Broadbent, Jamie Jellicoe
2	17/03/2016	Fincons contributions	Stefano Miccoli
3	18/03/2016	Fraunhofer contributions	Miggi Zwicklbauer
4	18/03/2016	IRT contributions	Fabian Schiller
5	21/03/2016	Review of contributions	Stefano Miccoli
6	24/03/2016	Added HbbTV validator to dev process	Stefano Miccoli
7	31/03/2016	Final release	Stefano Miccoli

Acknowledgement: The research leading to these results has received funding from the European Union's Horizon 2020 Programme (H2020-ICT-2015, call ICT-19-2015) under grant agreement n° 687921.

Executive Summary

This deliverable describes guidelines for developers within the MPAT project. Guidelines include: code style rules, guidelines for commenting code and use of developer tools for versioning, build and testing.

The process definition will follow practices of other successful open source projects like Chromium or Open Stack. Moreover, it will provide tools that support the collaborative software development like version control systems, bug tracker or build tools and define guidelines for the use of those tools

Table of Contents

1 Introduction

Partners agreed to adopt an agile approach for managing development of MPAT. As result of this decision, some fundamental concepts are borrowed from Scrum framework, starting from the product backlog which is defined by results from previous work packages, with particular relevance for the ones from task 4.2 - *System architecture design.*

Development activities are arranged in sprints, which duration is fixed and set in two weeks. Each sprint is divided in small tasks that ideally require no more than one day to be accomplished. Only one partner can be assigned to a single task.

Conference calls are scheduled between sprints. Typical agenda has two main points: last sprint review and next sprint planning. During the first one, partners review tasks that were completed and tasks that were not. During the sprint planning, priority of remaining product backlog items is discussed and upcoming items are assigned to partners. Each partner defines independently tasks related to its product backlog item.

1.1 Software development process

During development, code repository is structured at least in three different branches:

- Development branch, master branch in git repository, represents current status of MPAT, including all stable features.
- Staging branch allows developers to deploy in staging environment features and bug fixes that need to be tested by merging their branches in it.
- Production branch represents current status of production environment. Code versioned in this branch is supposed to be ready for production deployment anytime.

To minimize conflicts between partners' activities, development workflow should follow this steps:

1. At the beginning of work on a story, a dedicated branch, hereafter referred as story branch, is created starting from the development branch. In this way every story implementation uses current stable version as a starting point (Step A of Figure 1).
2. Developers involved in story implementation work on a local clone of the story branch. Ideally every single push to the remote story branch might not take place until automated tests are successfully executed in local environment, but the fact that story tasks are assigned to only one partner this condition could be considered optional (Step B of Figure 1).
3. Once story implementation is completed, story branch is merged in staging branch to be tested.
    a. In case of tests failure, fixes are implemented in story branch and merged again in staging branch.

b. If story concerns HbbTV front-end app features, when automated test suites are completed successfully, partners proceed with manual tests on selected devices and submission of pages involved in development to IRT HbbTV validator tool [1] to ensure that generated HTML is HbbTV compliant.

This step is repeated until all tests execute successfully. See Testing and Deployment sections for further information (Step C of Figure 1).

4. When all tests succeeded, story branch is merged in the development. The use of story branch prevents that other features present in staging environment, which might be not fully tested yet, are accidentally merged in development branch (Step D of Figure 1).

5. To keep code repository as clean as possible, after story development is completed related branch has to be deleted (Step D of Figure 1).

6. When stories are ready to be released, development branch is merged in production one, then annotated tag of production branch is created following guidelines of versioning best practices in this document (Step F of Figure 1).

7. At the end of each sprint, staging branch is aligned to development.

This management complies with MPAT scrum-like guidelines for partially completed stories. At most, every story is completed by the end of the sprint which means that all stories branches are already merged in development branch. In case a story is not completed, new story with remaining tasks is put in the product backlog. When this derived story is processed in next sprints, old story branch is used and merged again in staging branch.

Consequence of previous assumptions on status of production branch, merges of development branch in production branch have to take place only before deployment during scheduled release date (Step E of Figure 1 represents period between story completion and release date). This also gives the user who is assigned to the deployment one last chance to verify what is going to be released as a diff between development and production branches.

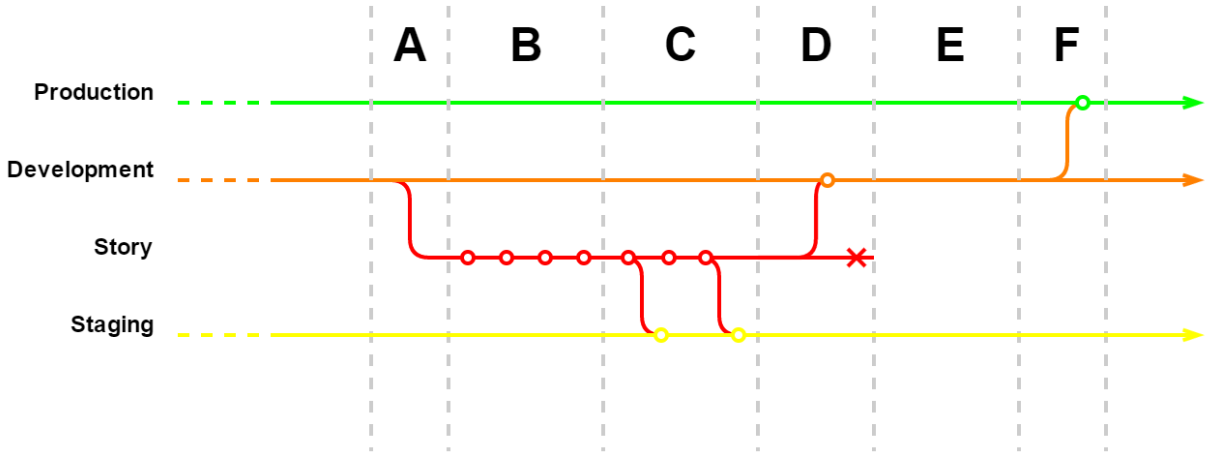

Figure 1 - Development workflow with branches

1.2 Requirements

The process of gathering the requirements for the MPAT project is derived from several sources:

- Scenarios (WP2)
- Consistent-user-experience design (WP3)
- System design and quality assurance (WP4)

The triggering requirements come from the set of scenarios developed in WP2. Those scenarios build the fundamental implications for WP4 and WP5 and were converted into more specific use cases, allowing a detailed description of the technical demands. As the project aims to develop a toolkit which consists of basic modules and over and above add-on extensions called plugin-ins, the first step is to carve out the basic functionality of the system by shaping out the synergies of different use cases and therefore clustering and defining modules which can be derived by those use cases to form this basic set of tools.

The second set of requirements comes from WP3 dealing with the user interface for both front- and backend. Again these will also be derived from the use cases developed in WP2. The relevant output for the technical specification will be the navigational concepts for the user interface design and the interaction design for the multiplatform applications.

An integral part of WP4 is to analyse external conditions which can be derived from a close look at the market. Task 4.1 focusses on a state-of-the-art analysis in regard to content-management-systems used nowadays in the web and their adaptability for the purpose of MPAT. The DoW already mentions the possibility to choose between an existing CMS such as WordPress or to build a system from scratch. It demands that not only technical issues are taken into consideration, but also business and usability issues lead to a final determination of the later used system. Further analysis focusses on the target platforms and end-devices as well as legacy systems, which have to be taken into account.

The above mentioned requirements are then validated by analysing the issues in regards to contra-dictions or conflicting goals. Finally, this leads to the description of the target system with functional and non-functional requirements, which are listed in a single document for further processing and building the architecture of MPAT.

1.3 Testing

To ensure code consistency, maintainability and overall quality, MPAT development adopt a test-driven [1] approach. In that way, as a first step during story process, developer writes a test which is expected to fail. Then he implements code, kept as simple as possible, to pass the test while satisfying functionality needs.

Each MPAT core module has to implement its own set of unit tests, based on PHPunit library while admin and content editor user interfaces are expected to implement unit tests for JavaScript functions using QUnit framework. On top of these two main parts of MPAT software, acceptance tests based on Codeception are written to ensure the achievement of stories requirements.

Due to the lack of any non-commercial solution for automated testing on real HbbTV devices, a different approach must be used in testing phase of stories which have impact on final user applications.

First, unit tests based on QUnit and acceptance tests written with Codeception are executed on standard browsers. Following versions of final user application are tested:

- Desktop version as seen in browser.
- Companion screen version relying on browsers device emulation feature.
- HbbTV version, relying on third-party firefox extension FireHbbTV.

Finally, partners test features in a selected set of devices, based on their HbbTV revision implementation. Once a stable set of front-end application features like templates and modules is defined, short list of use cases can be tested at periodic workshops organized by partners with broadcasters and developers.

Partner	Brand	Model
FOKUS	LG	55LM671S-ZB
FOKUS	LG	42LA8609
FOKUS	LG	47lb731v
FOKUS	LG	47LB650V
FOKUS	LG	UF8559
FOKUS	Loewe	50404W42
FOKUS	Panasonic	TX-L47ETW60
FOKUS	Panasonic	TX-47ASW654
FOKUS	Panasonic	TX-55CXW754
FOKUS	Philips	40PFL80085/12
FOKUS	Philips	48PFS8209
FOKUS	Philips	55PUS9109/12
FOKUS	Samsung	UE40D8090
FOKUS	Samsung	UE40C8790
FOKUS	Samsung	UE40ES8090S
FOKUS	Samsung	UE46F8090
FOKUS	Samsung	UE46F8090SL
FOKUS	Samsung	UE55H6470SS
FOKUS	Samsung	UE65HU7590LXZG
FOKUS	Samsung	UE55H8090SVXZG
FOKUS	Samsung	UE46H7000
FOKUS	Samsung	UE48JS9090Q
FOKUS	Sharp	LC-50LE752E
FOKUS	Sony	46W905A

Partner	Brand	Model
FOKUS	Sony	50W815B
FOKUS	Sony	55W955B
FOKUS	Sony	KD55X9305C
FOKUS	TechniSat	ISIO 42
FOKUS	Toshiba	46WL863
FOKUS	Toshiba	42M7463D
ULANC	Panasonic	TX-40CX680B
ULANC	Humax	FVP-4000T
RBB	Samsung	UE40F6500
RBB	Humax	Icord Cable
RBB	Xoro	HRK 8910 Hbb+
RBB	Panasonic	TX-L42EW6W
RBB	Grundig	37VLE973 BL
RBB	Inverto Volksbox	IDL 6651N
RBB	Inverto Volksbox	IDL 6654N
RBB	Sony	KDL 22EX555
Fincons	Panasonic	TX42AS650E

Table 1 - List of available HbbTV devices in partners' lab

1.3.1 Defect tracking

During development, defects might occur. Their fix should follow these rules:

- Trivial defects are fixed during current sprint without specific tracking
- Critical and blocker defects generate specific task in sprint backlog, which is discussed in next bi-weekly conference call, and are fixed during current sprint. Time consuming defects might that other tasks have to be moved back to product backlog.
- Time consuming but non severe defects are tracked in backlog to be fixed in future sprints

Tracked defects, whether they were fixed during last sprint or they still have to be fixed, are discussed during next bi-weekly telco. In the first case, partners review causes and solutions adopted to improve future developments. In the other case, Product Owner set the priority of defect task, deciding if it have to be fixed during next sprint or future ones.

1.4 Deployment

All deployment tasks, including generation of packages, validation of final user application, distribution of assets to CDNs are handled by the deployer tool, part of MPAT core, with peculiarities for each environment.

Environment structure reflects branches definition in code repository. There are two remote environments, staging and production. Regarding development branch there is no need of a remote environment, but developer have their own local setup based on it.

Deployment of MPAT application in staging environment is triggered by the automation server after each push in remote staging branch.

Deployment of MPAT application in production environment is achieved only by manual interaction in automation server user interface, in the form of execution of predefined task, to prevent release of bugs and incomplete features which might be in the production branch as a consequence of an accidental merge.

2 Supporting tools

2.1 Technical docs and repository

The technical docs and the repository will be stored at GitLab deployed at a server of Fraunhofer FOKUS https://gitlab.fokus.fraunhofer.de/MPAT/MPAT (Figure 2).

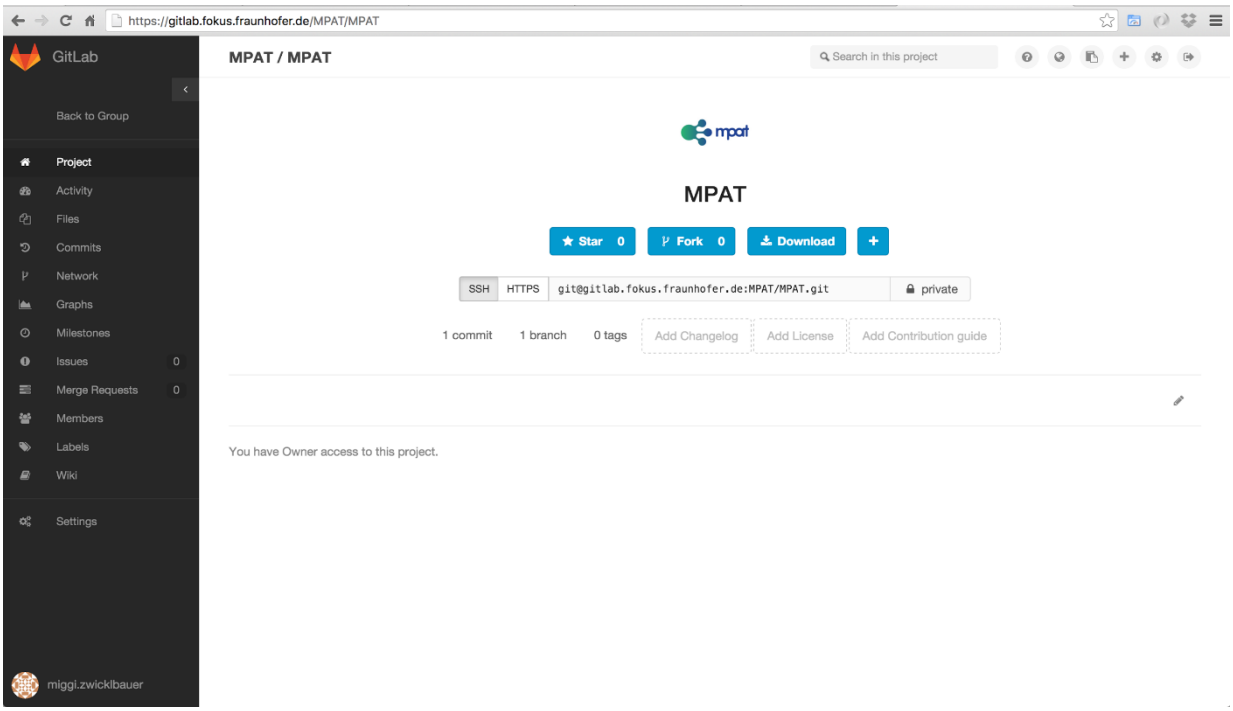

Figure 2 - GitLab service

GitLab is a management system for git repositories [2] that runs on the FOKUS infrastructure. It provides code reviews, issue tracking, a wiki and other features we plan to use for the development phase in MPAT. Each partner of WP5 – *Implementation* got their own account in the tool to work on the same code. GitLab also provides the feature of discussing issues, do code reviews and make inline comments. The web service does also have the integration of JIRA, which will be used to plan two week sprints including the creation of tasks. Also the integration of Jenkins is provided for the testing.

By having a stable and basic code from MPAT we are planning to publish the code also on GitHub. GitHub [3] has a huge community and is one of the most popular repositories.

2.2 Version control system

For the version control of the MPAT development we are using GIT. The version control system is free software distributed under the terms of the GNU General Public License version 2. The latest source release of GIT is 2.7.3 (10th March 2016). The open source SCM handles multiple developers building multiple branches which can be entirely independent of each other.

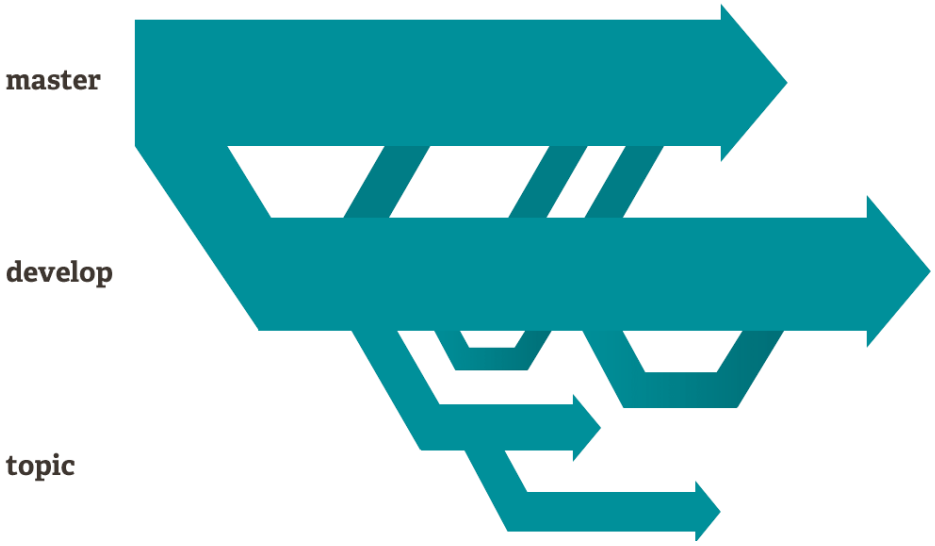

Figure 3 - GIT workflow using branches [15]

In direct comparison to the other popular version control system SVN, GIT is faster in committing, updating and blaming [4].

2.3 Continuous integration manager

In the MPAT project we are planning to use Jenkins [5] as the continuous integration manager. Jenkins is an open source cross-platform continuous integration and continuous delivery application. We plan to use Jenkins to build and test MPAT. This will be making it easier for MPAT developers to integrate changes to the project and for users to obtain new builds. Also Jenkins will be used to integrate a number of tests to bring MPAT to stable and productive tool.

Jenkins brings a lot of features with it. It is running on OS X, Linux and Windows, which means that builds and test loads can be distributed to multiple computers. The Jenkins community provides also a huge range of excising plugins that can be used. It is also possible to write own (MPAT specific) plugins to generate tests on e.g. HbbTVs.

2.4 Unit test suite

As MPAT is based on WordPress and got a codebase of PHP we are planning to using PHPUnit [6] for the MPAT core functionalities. PHPUnit is an instance of the xUnit architecture for unit testing framework. PHPUnit can be also integrate in Jenkins for the testing of MPAT. For the Content Editor and Admin UI functionalities we are planning to use QUnit [7]. Also the Frontend application functionalities (e.g. user interactions in modules) will be tested with QUnit tests. QUnit is a JavaScript Unit testing framework that can be used for any generic JavaScript code.

Both testing frameworks are open source.

2.5 Acceptance and functional tests suite for MPAT backend

As said before, MPAT got two different parts to test:

- The Backend / Editor UI
- Frontend generated applications

Both components will be tested in acceptance and functional tests. The test suite for this test will be a combination of Codeception and the Selenium WebDriver. Codeception is an open source testing tool to test all links, forms, pages, APIs, and anything else in just minutes using automated test Codeception scripts [8]. It provides also an integration into Selenium.

“Selenium automates browsers” the website says [9]. The Selenium WebDriver provides a collection of language specific buildings to drive a browser. It can be used to create robust, browser-based regression automation suites and tests, as well scale and distribute scripts across many environments.

2.6 Deployment

The automatically deployment will be done by Capistrano [10]. Capistrano is a remote server automation tool that run with Ruby. It allows to copy code from the MPAT repository with GIT to the server via SSH. Capistrano will be responsible to perform pre and post-deploy functions like renaming files, running DB migrations or restarting the webserver.

2.7 Bug tracking

A bug tracking system is a necessary component of professional software development.

In the development of MPAT bugs will be tracked in the GitLab at Fraunhofer FOKUS. The tracked bugs will be available under the Issues tab that linked directly to the project directory in JIRA. New bugs will be also stored in JIRA which is used for reporting of bugs.

3 Best practices

3.1 Code style rules

Apply common sense. If adhering to these rules compromises readability, then they need only be applied as far as is practicable. Based on the Mozilla Coding Style Guidelines [11] and *The Elements of Programming Style*.

3.1.1 Whitespace

No tabs. No whitespace at the end of a line. Unix-style linebreaks (`'\n'`), not Windows-style (`'\r\n'`).

End each file with a single newline.

3.1.2 Line length

80 characters or less (for laptop side-by-side diffing and two-window tiling).

3.1.3 Identation

Four spaces per logic level.

3.1.4 Control structures

Use K&R bracing style [12]: left brace at end of first line, cuddle else on both sides.

Always brace controlled statements, even a single-line consequent of an if else else. This is redundant typically, but it avoids dangling `else` bugs, so it's safer at scale than fine-tuning.

Break long conditions after `&&` and `||` logical connectives.

Don't put an `else` right after a `return`. Delete the `else`, it's unnecessary and increases indentation level.

3.1.5 Licensing

Ensure that any required licensing information, such as boilerplate declarations at the top of files, are included.

3.1.6 Variables

Declare local variables as near to their use as possible. Avoid overuse of temporary variables. Name variables so that they won't be confused. Name variables descriptively.

3.1.7 Functions

Name functions after their purpose. Replace repetitive expressions by calls to common functions. Modularise where possible; functions should not be long. This aids testing and readability.

3.1.8 Debug

Don't leave debugging statements in code when submitting to the central repository.

3.2 Code documentation

Use JavaDoc-style comments [13]. Most documentation should be auto-generated, as with PHPDoc, to minimise the effort required for the documentation process. Class and method level comments should be brief and succinct, whilst descriptive where complex code is present.

Code should almost be documented by the code itself; writing easy-to-read code with descriptive variable and function names goes a long way to making code maintainable.

3.3 Versioning

Use Semantic Versioning [14]. Given a version number MAJOR.MINOR.PATCH, increment the MAJOR version when you make incompatible API changes, MINOR version when you add functionality in a backwards-compatible manner, and PATCH version when you make backwards-compatible bug fixes.

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

3.4 Unit test

Test coverage should ideally be 100%, but 95% is acceptable.

Tests are implemented by functions that are marked as test cases in some way, and grouped together into testing suites. Each test suite defines a setup function and a teardown function, which are run before and after the test, respectively.

Each test function declares an expected outcome. These are then checked using specific assertions. If one of the fails or is not met, then the test has failed. If all assertions are correct then the test has passed. These assertions are derived from a set of software requirements, formed before development takes place.

As with any other source code, these tests must be readable and understandable.

Test Driven Development (TDD or regression testing) is a good way to ensure progress towards a goal. As new functionality is developed, and the test suite is added to and run again, all of the previous code is re-tested. This ensures that the new code does not affect the integrity of what has already been written.

3.5 Logging

Four logging levels are to be used:

Debug

This is the highest verbosity logging level and should only be used for development and testing. The information displayed at this level of logging will likely not be seen by the end user.

Information

The logging level is used to convey information helpful for the running and management of the system. However, items included at the Information level, do not indicate a problem or a potential issue with the running system.

Warning

Messages at the Warning level indicate that exception or errors have been encountered, but they have been handled by the application. The application can continue to operate normally.

Error

These messages are for uncaught or unhandled exceptions that typically result in the application exiting.

Glossary

WP	Work Package
T	Task
DoW	Description of Work
UI	User interface
CMS	Content management system
CDN	Content delivery network
SCM	Source control management
API	Application programming interface
TDD	Test driven development

Partner Short Names

Short Name	Name
FRAUNHOFER	Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V. (DE)
IRT	Institut für Rundfunktechnik GmbH (DE)
RBB	Rundfunk Berlin-Brandenburg (DE)
ULANC	Lancaster University (UK)
MEDIASET	Reti Televisive Italiane S.p.A. (IT)
LEADIN	Leadin Oy (FI)
FINCONS	Fincons S.p.A. (IT)
IMT	Institut Mines-Telecom (FR)

References

[1] «IRT HbbTV validator,» [Online]. Available: http://hbbtv-live.irt.de/validator/.

[2] “Test driven development,” [Online]. Available: https://en.wikipedia.org/wiki/Test-driven_development.

[3] «GitLab,» [Online]. Available: https://about.gitlab.com/.

[4] «GitHub,» [Online]. Available: https://github.com/.

[5] «Git about,» [Online]. Available: https://git-scm.com/about/small-and-fast.

[6] «Jenkins,» [Online]. Available: http://jenkins-ci.org/.

[7] «PHPunit,» [Online]. Available: https://phpunit.de/documentation.html.

[8] «QUnit,» [Online]. Available: https://qunitjs.com/.

[9] «Codeception,» [Online]. Available: http://codeception.com/.

[10] «Selenium,» [Online]. Available: http://www.seleniumhq.org/.

[11] «Capistrano,» [Online]. Available: http://capistranorb.com/ .

[12] «Mozilla coding style guidelines,» [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Coding_Style.

[13] «K&R bracing style,» [Online]. Available: https://en.wikipedia.org/wiki/Indent_style#K.26R_style.

[14] «Javadoc style comments,» [Online]. Available: http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html.

[15] «Semantic versioning,» [Online]. Available: http://semver.org/.

[16] «GIT,» [Online]. Available: https://git-scm.com/about.

List of Figures

List of Tables