RWTH Aachen University

Hightech Entrepreneurship and New Media

Definition of Done

Alexandra Wörner Dominik Studer Frederik Zwilling
Dev Sharma Ali Can Demiralp

January 8, 2015

Contents

1 General **3**

2 Git Usage **3**

2.1 Branching . . . 3

2.2 Commit . . . 3

2.2.1 Commit early and often . . . 3

2.2.2 Describing Commits . . . 3

2.3 Push . . . 4

2.4 Merging/Pull-Requests . . . 4

2.5 Tips and Tricks . . . 5

2.6 Dos and Don'ts . . . 6

3 Programming Conventions **6**

3.1 JavaScript . . . 6

3.2 HTML / CSS . . . 6

3.3 PHP / MySQL . . . 7

3.4 Development Environment . . . 7

4 Testing Process **8**

4.1 System Tests . . . 8

4.1.1 General Workflow . . . 8

4.1.2 Issue Schema . . . 9

4.1.3 Test Case Schema . . . 9

4.1.4 Test Log Schema . . . 9

4.1.5 Debugging / performance measurement tools . . . 9

4.2 Automated Tests . . . 10

4.2.1 Back-end . . . 10

4.2.2 Front-end . . . 10

4.2.3 Database . . . 10

5 Gantt Chart **12**

5.1 Development 12

5.2 Conventions 12

6 Technologies **13**

6.1 Mesh Processing 13

6.2 RoleSDK 13

6.3 XAMPP 13

6.4 X3DOM 13

6.5 Downsampling 13

6.6 Format conversion 14

1 General

To declare a feature as ”done”, it has to be fully implemented and working. All related subtasks as well as the user story/stories chosen for the feature have to be covered. The code has to be pushed, reviewed and merged with the develop(ment)-branch of the project.
The following sections describe was has to be done or provided in terms of code style, testing and documentation.

2 Git Usage

2.1 Branching

Proper git-based development advices to develop each (independent) feature in a separate topic branch. This enables to switch between different working contexts very easily.
To create such a branch and to check it out to start working you do:

```
git checkout -b your-branch-name
```

where *your-branch-name* needs to be replaced appropriately. This creates a new branch based on the one currently checked out in your working copy.
To start from a different start point, e.g. from the development branch, to start a new feature do

```
git checkout -b your-branch-name development
```

2.2 Commit

2.2.1 Commit early and often

Once you have achieved reasonable progress, you might want to commit the changes. To do so, you need to tell git what files should be included in the commit by adding them to a staging area. This area is meant to give you a chance to properly compose your commit, possibly even taking only a few changes but not all of a single file, e.g. if it makes sense to separate them. Add files like the following

```
git add a-modified-file.h a-newly-created-file.cpp
```

2.2.2 Describing Commits

To make is easier to use git tools we have a specific format we expect from commit messages. It is:

Component: Very short description of change
Longer text describing the change.

The subject is a brief explanation of the change and may not exceed 60 characters in length. It should start with a component name, e.g. the name of a plugin the code belongs to. The subject line has a specific relevance. Many git tools (e.g. gitk) show only the first line in the default view and therefore this line should be short and concise. Then follows an empty line and then a longer text describing the change in more detail. If you think you cannot fit the change in your subject chance is that you should break up the commits in more smaller commits. The longer text can be omitted if the subject line is already sufficiently clear. In the longer message think that you need to explain the change to someone else.
An example:

overview: added thumbnails of models
Added table with model thumbnail and links to the models. Implemented access to database to get the thumbnail-url.

2.3 Push

So far, you have only committed to your own branch in your own (local) clone of the repository. To share your development with others, you should push your branch to another repository. We have central repositories at *http://eiche.informatik.rwth-aachen.de:4080/henm1415g2* that we use to share code with each other. The process of copying your changes to another repository is called pushing. To push your local branch do:

```
git push origin name-of-local-branch:name-of-remote-branch
```

If you push with the option *–set-upstream*, you can just use

```
git push
```

for future purposes.

2.4 Merging/Pull-Requests

Merging is the process of integrating a branch into another one. There are two main scenarios where a merge is required: merging a topic branch into the development branch, or merging many topic branches into a temporary merge branch (often called "current").
If you want to merge your branch into the development branch, create a pull-request. Another developer then merges and reviews the branch. After checking if everything still works, the reviewer can push the updated development

branch.
You can merge changes with:

```
git merge topic—branch
```

This assumes that the branch in which to merge the topic branch is currently checked out, i.e. the branch named ”development”, is checked out.
If we have a stable development version (e.g. after a sprint), we can merge the development branch into the master branch.

2.5 Tips and Tricks

To add specific parts that you changed but not all use the -p parameter like so:

```
git add —p a—modified—file.h a—newly—created—file.cpp
```

This will bring up an interactive mode to stage only specific changes.

Delete old local branches from remote with:

```
git fetch ——prune
```

You can put changes on hold for later, for example to clean up the tree or to switch to a different topic branch to work on a different subject. To stash them away do

```
git stash
\# Or alternatively with a useful description:
git stash save ”unfinished code for ...”
```

You can list and view those stashes with

```
git stash list
git stash show
```

To retrieve a stash you have two options. The apply sub-command will retrieve the stash contents and apply them to the current state of the working copy. The pop sub-command will additionally drop the stash if it was successful in applying it. The drop command drops the stash without applying it.

```
git stash apply
git stash drop
git stash pop
```

The stashes are organized as a stack. Without a parameter, all stash commands operate on the most recent stash.

2.6 Dos and Don'ts

Here, you can find a list of hints what you should do or should not do. We created this list from the mistakes we learned from in the past to avoid problems in the future.

- You shouldn't upload huge files into the repository to keep git fast and stable and be able to compare versions.

- You shouldn't copy files to modify them, this makes merging time-consuming and risky. If you want to change something without modifying the stuff of others directly, you can just do it in a new branch.

- You should be careful if you work on different branches, so you do not accidentally merge two feature branches together. This can create huge merge conflicts.

- You should follow the commit messages conventions of this DoD to make it much easier to understand and find commits afterwards.

- You shouldn't push directly into develop.

Source: trac.fawkesrobotics.org/wiki/GitNotes

3 Programming Conventions

3.1 JavaScript

There are many coding and style guidelines for JavaScript available on the internet. Upon comparatively surveying these guides, we decided to follow *Google JavaScript Style Guide*. This guide has great coverage, including language and documentation rules, and provides justifications for each rule it contains. The guide is available here.

3.2 HTML / CSS

All front-end code will be written in HTML5 and CSS3. Upon surveying online guidelines for modern HTML, we decided to closely follow *Google HTML / CSS Style Guide*. The guide is available here.

3.3 PHP / MySQL

The back-end is written mainly in PHP and MySQL. After taking different style guidelines into consideration, we decided to follow this coding standard or PHP and the guidelines specified here. The coding conventions used for MySQL can be found here.

3.4 Development Environment

Because we mainly use HTML, JavaScript and CSS, we agreed that everyone uses his favourite IDE. There shouldn't be IDE specific differences and this way everybody can work with the tools he performs best with.

4 Testing Process

4.1 System Tests

4.1.1 General Workflow

1. Derive test cases
   If the tester is not the same person as the programmer of the code to test, the tester first checks the commit message of the code. There it should be stated, which task (of the sprint backlog) should be fulfilled by the code (otherwise the tester may ask the developer directly). The tester accesses the task description in the sprint backlog in the gitlab repository.
   If the programmer does the testing, he/she may derive test cases during or before the programming activities.
   After acquiring the task description, the tester specifies one or several test cases checking that the intended functionality is fulfilled. The tester uses his/her experience to add additional test cases for defects that are likely to exist (error guessing). For important parts of the software, equivalence partitioning and boundary value analysis can be used to derive further test cases.
   Test cases are written in a format described below. If tests have to be executed in a certain order, the order is noted down as well.

2. Execute test
   For each test case, the tester prepares the testing environment accordingly. Afterwards the tester executes the steps listed in the test case description and evaluates the result(s). If the result differs from the one specified in the test case description, a programmer may directly start debugging. If code has been changed during debugging, all tests are repeated. It may be efficient to go on with tests that have not been executed yet before repeating tests (as its more likely to find errors there and all tests would have to be repeated again).

3. Test logging
   The tester notes in the test log, which tests passed and which did not. For each test, that did not pass, he/she creates an issue using gitlab. Programmers do not create issues, if they already debugged the failure.

4.1.2 Issue Schema

- The title of the issue should be test case id and date of test execution
- The description should contain who did the test and what was the difference found in comparison to the expected result
- The issue is assigned to the programmer of the code that had been tested

4.1.3 Test Case Schema

Each test case should at least contain the following:

- Some unique name/id, such that it can be referenced in the test log
- The input data to be used by the tester
- The steps to be done by the tester
- The expected output/result to be compared to the actual result the tester sees during execution
- OPTIONAL: Some precondition, that has to be fulfilled before executing the test

4.1.4 Test Log Schema

Each test log entry contains:

- The date, when the tests have been executed
- The name of the tester
- The ids/names of the tests that have been executed
- The result of the testing (OK/NOK). If some tests failed, the tester creates two test log entries (one for successful tests and the other one for failed tests)

4.1.5 Debugging / performance measurement tools

Firefox:	built-in (Firebug)
Google Chrome:	built-in
Internet Explorer:	built-in (F12)
Safari (Mobile):	Remote debugging
Android browser:	Remote debugging (with Chrome)
Opera:	built-in (Dragonfly)

4.2 Automated Tests

We use automated tests to ensure correct behaviour of functional components. We separate the automated tests into Back-end-tests with Jasmine and Front-end-tests with Selenium. Interaction with the MySQL database is tested with PHPUnit.

4.2.1 Back-end

To test JavaScript code in the Back-end, we use Jasmine together with the plugin jasmine-jquery. The Jasmine Website and a tutorial with many examples can be found here.
The tests are located in *src/tests/jasmine/spec* and are written as JavaScript files. The tests and the JavaScript files containing the code which should be tested have to be included in *src/tests/jasmine/SpecRunner.php*. You can run tests by just opening *src/tests/jasmine/SpecRunner.php* in the browser. This also allows us to test different devices and browsers very simply.
If you want to test JavaScript code that interacts with website elements that are missing in the test scenario, you can use the spyOn() function from Jasmine which allows you to replace the output or register calls of any JavaScript function. Another possibility is to include those elements as fixtures, small files only containing the desired element. These elements can be loaded in the test suite and tested with the help of jasmine-jquery. The plugin also allows access to already existing DOM elements. To work with them, call them using the common jQuery syntax and apply the methods Jasmine provides or custom methods.

4.2.2 Front-end

To test the Front-end and user interaction automatically, we use the tool Selenium. The website where you can download it can be found here. If you finished a feature of user interaction, you can add tests by recording your user interaction in the browser. Selenium can then replay your actions and check if it is able to redo everything. You can also add additional assertion steps to check for some condition at any point in the sequence of recorded actions. The test cases and the test suite are saved in *src/tests/selenium*. Please also update the test suite to include your test case.

4.2.3 Database

Database interaction is tested with the tool PHPUnit which you can find here. Test suites, which are classes extending the *PHPUnit_Extensions_Database_TestCase* class. A test suite contains one

or several test case(s), public function containing assertions.
A single test suite is executed on the console with the command

```
php <path-to-phpunit.phar> [--verbose] --configuration
    <configuration-file>.xml <test-suite>
```

from inside the directory in which the test suite's file is saved. Thereby, <configuration-file>.xml contains the information for the database access, i.e. the database name, host, user name and password. It can also contain the directory and/or the files of the test suites to be executed. In that case, use the above command without specifying a test suite.
More details to the test classes and assertions can be founc in the PHPUnit online manual.

5 Gantt Chart

A Gantt chart is a bar chart that shows the tasks of a project, when each task must take place and how long each will take. It is helpful in monitoring project tasks within a project, especially which task has to be done first in order to start with the next one.

5.1 Development

Before creating the chart, identify the tasks needed to complete the project. Key milestones may be helpful in structuring ... Then identify the time which is required for each task. At last, write down the order: Which tasks must be finished before another, on the first task dependent task, can begin? Which can be worked on at the same time? And which tasks must be completed for achieving a milestone/before the next review?
Afterwards, create the chart by writing down the tasks in the previously defined order. For each task, draw a bar spanning the required time. Subsequently, assign team members to the tasks. Make sure that everyone has approximately the same workload.

5.2 Conventions

- Try to keep it simple, i.e. choose not more task than fit onto one page.
- Highlight milestones/reviews
- Assign different colours to team members or a group working in the same area, respectively
- Indicate the (temporal) dependency of one task from another with arrows

6 Technologies

6.1 Mesh Processing

The 3D Model used for collaborative viewing on the web is captured using 3D scanning using Breukmann scanSCAN and stitched together using Optocat software. The mesh obtained contains millions of triangles and have to be cleaned and down sampled before it could be used directly for the web.

6.2 RoleSDK

ROLE (http://www.role-project.eu/) provides a learning environment in the browser which can be personalized with widgets and shared with others. We use the RoleSDK to embed our product into the already existing learning ecosystem. It provides us possibilities for inter-device and inter-widget communication.

6.3 XAMPP

XAMPP is a open, cross-platform web server solution stack. We mainly use it to test our project in small scale because it is easy to set up and provides many features we might need. We use the Apache HTTP server and MySQL as database.

6.4 X3DOM

X3DOM is an open-source framework and runtime for 3D graphics on the Web via WebGL. It can be freely used for non-commercial and commercial purposes, and is dual-licensed under MIT and GPL license. It provides us the ability to render any 3-D scene on the web without any plug-in.

6.5 Downsampling

For down sampling of the mesh, we use Meshlab which is an open source software.
The software is available here.

Optocat exports the 3D mesh as an STL/PLY file which can be easily imported by Meshlab for processing. In Meshlab we use filter *"Quadratic Edge Collapse Decimation"* for reducing the polygon count. It is also used for general cleaning of the 3D mesh.

6.6 Format conversion

Meshlab exports the down sampled and cleaned mesh as *.WRL* format which is a file extension for a *Virtual Reality Modeling Language (VRML).* We have to convert it into *X3D* file extension that is used in X3DOM which is an open-source framework and runtime for 3D graphics on the Web. We use InstantReality for this. The online conversion tool is available here.

InstantReality is an online translator between *Classic VRML or XML (.WRL)* encoding and can convert into X3DOM outputs (HTML5 or XHTML5). We just paste the *.WRL or .X3D* file in it and it creates a HTML output.
It could also be used offline in the similar way as Instant Reality installations include *aopt*, a local command-line converter.