Methodology

The FenixEdu Way

We been developing software for over a decade now, and over the course of all these years we've been fine tuning our own process. As the team grew past our core developers, we adopted a set of methodologies to smooth and coordinate our software development. Such methodologies spawn the software development process concerning issues about version control, branching model, versioning of software artefacts, and contribution flows to such software artefacts.

Although some of these rules may seem a bit overbearing, they are in place to make life easier for you and us. Trust us on this one.

Branching Model

The guidelines presented here are to be applied to every repository under the FenixEdu umbrella.

The main branch is the `master` branch. All commits to this branch **must** be tested and stable, to ensure that the `master` branch is in a releaseable state at all times.

All development should be done in feature branches, which should be kept in Forks and never pushed into the main repository. Once the development of a feature is complete, it should be integrated into the `master` branch.

Releases

Whenever a new version is to be released, a new branch called `release/x.y.z` should be created from the `master` branch. In this branch, the version of the project is to be bumped in a single commit, and this commit should be tagged with a tag named `vx.y.z` (e.g. v2.2.1). If a new development version is desirable, a version bump (from one SNAPSHOT to another) commit should be created in the `master` branch. In no occasion should the release commit be reintegrated into the `master` branch.

For more information, see this presentation.

Old Branching Model - GitFlow

According to our best practices, all modifications to the code base must be associated with a Github issue.

To handle such issues, we adopted GitFlow, which help us to adapt, organize and manage our software development best practices from Git features. Although we follow Gitflow's branching model, we do not enforce the use of the Gitflow tool. We believe the developers are accountable for their actions, and as such, they do not need tools to enforce the usage of certain rules while using a particular version control system. Using the Git tool directly while using this branching model approach demonstrates that the developer understands correctly the strategy, while having all the Git version control system capabilities.

Versioning

FenixEdu is composed by a large set of code libraries, which built on one another. Hence, we need to take versioning very seriously or else, a simple code change might break the code.

To version our software, we consider the Semantic Versioning approach. The Semantic Versioning approach comprehends a version format composed by three numbers X.Y.Z, where each number increment is associated to a set of rules. These rules allow us to quickly identify if we can upgrade a dependency safely without breaking our code.

Building

To build our software, we use Maven. Maven allow us to declare both the build and release lifecycles of our projects. To ease such configuration, we have pre-configure four parent pom projects from which your pom can inherit these configurations from.

Contributions

As said before, FenixEdu is composed by a large set of projects that need to be properly managed. As such, we assign a main team of developers to each project. Nevertheless, developers outside of that main team can still contribute to the projects via the Github Pull Request feature. While reviewing such pull requests, each main development team also acts as the quality assurance team for that project. To contribute or request a change to a particular project, the developer must:

1. Explain the rationale for the contribution or change
2. If the contribution is accepted the developer must fork the project
3. The developer commits the discussed and agreed changes to his fork
4. The developer submits a pull-request to merge the respective modifications

Code Style

All code written for FenixEdu should be formatted with the same code style. This is to simplify the reintegration of code between different developers. You can find the code style specification here:

https://raw.githubusercontent.com/FenixEdu/fenixedu-java-codestyle/master/EclipseFenixCodeSyle.xml

This the of code style specification is supported both by Eclipse and InteliJ

Code Conventions

Package names are always lowercase, invalid characters are simply stripped down. All project's java resources are locate in packages under: `<organization>.<project>.<module>`, or `<organization>.<project>` for single module projects. For example, the core module of the bennu project, that is part of the fenixedu organization, has a base package name: `org.fenixedu.bennu.core`.

Inside the base package, resources are organized as follows:

Package	Resource Types	Depends on
`<base>`	Configuration managers	-
`<base>.domain`	Domain entities and related objects	`<base>`
`<base>.service`	Services	`<base>, <base>.domain`
`<base>.api`	Rest APIs	`<base>, <base>.domain, <base>.service`
`<base>.api.json`	JSON Adapters for the Rest API	`<base>, <base>.domain, <base>.service`
`<base>.ui`	Controllers, Portal Applications and Functionalities, and all other UI related resources	`<base>, <base>.domain, <base>.service`
`<base>.servlet`	Servlets, Initializers, Filters	`<base>, <base>.domain, <base>.service`
`<base>.task`	Scheduller Tasks	`<base>, <base>.domain, <base>.service`
`<base>.bootstrap`	Bennu Bootstrappers	`<base>, <base>.domain, <base>.service`

If the result includes duplications, like `org.fenixedu.scholar.ui.ui` (because the project scholar has a submodule with only the UI parts), just collapse into: `org.fenixedu.scholar.ui`.