Project Plan

It is important that all agreements between us, the client and the coach concerning the content, goals and methods of the project are set out in writing. The purpose of this document therefore is to describe these items and to formally agree upon how the project is realised. If anything changes over the course of the project, these changes need to be documented in written form. While an e-mail describing these changes is often sufficient, we propose that this document is updated accordingly (perhaps simply by referring to said emails) such that this document is the one document required to track our progress.

Project goal

From Spoofax's website:

> Spoofax is a platform for developing textual domain-specific languages with full-featured Eclipse editor plugins.

A feature that Spoofax is lacking is a Read-Eval-Print Loop (REPL) service generator. A REPL is an interactive programming environment that takes single expressions, evaluates them and prints the result(s). REPLs are a popular tool for programming because they facilitate exploratory programming and debugging. Common examples include command-line shells such as Bash and Python's REPL.

The deliverable for this project, then, is to create such a REPL generator for the Spoofax Language Workbench.

The final product

The final product will have to meet the following requirements and demands, as specified following the MoSCoW method:

- Must-have
  - Interactive shell
  - Works with any language defined in Spoofax
    - Optionally recognizes language-specific REPL commands defined in an esv file
  - Input & output history
  - Multiline input editing
  - Error reporting
  - Syntax checked expressions
  - Syntax highlighting
  - Integration with Eclipse
- Should-have
  - Ability to redefine identifiers

    - Environment inspection
    - Save and load shell state
- Could-have
    - Context-sensitive code completion
    - Hover over variables to see value, type and others
    - Literate programming
    - Integration with other IDEs (IntelliJ)
- Won't-have
    - GDB-style debugging and nested REPLs

If the above turns out to be (too) easy, the REPL can be extended into a language playground such as the one offered by the Swift programming language. If we decide to extend the project, new agreements will be made between us and the client.

Copyright for the final product will belong to TU Delft. It will be distributed under the Apache license version 2.0.

Methodologies and tooling

During this project we will work using the Scrum methodology with biweekly sprints. Our backlog and current- and completed sprints will be managed using Trello. Trello will also be used to communicate documents to the client and the TU coach. Sprints will end on friday, coinciding with most deadlines according to the planning for this project as mandated by TU Delft.

The Scrum methodology defines several roles to be attributed to members of the team. During the development of this project, the roles are divided as follows:

- Product owner: Gerlof
- Scrum master: Jente

Each sprint will start with a sprint planning meeting (friday after the reflection of the previous sprint, see below), in which user stories from the backlog will be assigned to project members based on their prioritization and their feasibility for this sprint. The created sprint plan is then discussed with the TU coach in the weekly meeting on monday at 11.00, where it might be slightly adjusted if the TU coach spots any issues.

Sprints end on fridays with a short (10-15 minutes) meeting with the client in which this sprint's progress is demoed. After this meeting, we will meet privately for the sprint reflection, where we discuss to what degree we have achieved our sprint planning and how we can improve on that. As mentioned in the previous paragraph, we follow this meeting with the planning of the next sprint.

During sprints, we will hold a daily scrum meeting in which we will discuss our progress of the previous day and our planning for today, as well as any issues that could be detrimental to the sprint goal.

We will use a pull-based development model using GitHub. GitHub Issues will be used to track issues (all of this will be linked to Trello in one way or another). To facilitate this, we will use repositories from our GitHub organization spoofax-shell. Each of us will then fork these repositories to our private GitHub accounts. We will also use one repository to hold all the documents (e.g. this document, the research project, et cetera) supporting our project.

A pull request will be designated a status tag **WIP**, **RFC** or **RDY**:

- WIP - Work In Progress: a PR tagged WIP is there to gather feedback; it will still change so it should not be reviewed in-depth.
- RFC - Ready for Comments: a PR tagged RFC is done as far as the implementor is concerned: the functionality is there, it is tested, documented and only a (thorough) review is left. If there are comments that cannot be resolved within reasonable time, the PR should be demoted to WIP. Before changing a PR to RFC, all test cases should pass on TravisCI, the continuous integration platform.
- RDY - Ready: a PR tagged RDY has been reviewed by at least one other reviewer, who deems it ready for merge. It is left open a short while to give the remaining team member time to review, after which it is merged.

This process ensures that all code has been reviewed and tested before being merged into the final product, thereby always ensuring a working product. By using continuous integration we aim to reduce code regression.

Since Spoofax is an already existing project, we will reuse most tooling that is already being used. This means that we will use the Java programming language, Maven for the build environment and JUnit for unit tests. Dependencies used by Spoofax (such as Guice and RxJava) will be used if needed. Additional tools to assist in maintaining quality code include TravisCI for continuous integration, Checkstyle for a coherent code style and FindBugs and PMD for static analysis.

Quality assurance

There are two types of quality to be assured:

1. Quality of the code;
2. Quality of the end product.

Quality of the code

The quality of the code will be assured by applying proper software development practices, such as unit testing, continuous integration, code reviews and static analysis. We will strive to uphold a clear and fitting architecture supported by proper application of the (correct) design patterns, resulting in low coupling and high cohesion. For more information on how our development is organised, please see the CONTRIBUTING file in our code repositories.

From this file, the definition of done can be extracted which we explicitly repeat here. An item is considered done, when:

- The feature as explained in the user story has been implemented;
- The code has been properly tested, with both unit and integration tests;
- The code has been properly documented;
- All other tests (TravisCI, FindBugs, PMD, Checkstyle) also pass.

Quality of the end product

The quality of the product will be assured by asking regularly for feedback from the clients. This will be done by discussing status updates, preferably with working demos to illustrate and support the progress.

Further towards the end of the development period, we will ask one or two members of the client's research group to use the REPL for one of the existing (simple) languages implemented in Spoofax, perhaps while slightly changing the language to see how a REPL assists with this task. This is to gather feedback from someone who has not been involved in the development but who will be an end user. Results from these tests will then be discussed with the client to see where the product still needs improvement. This way, we hope to deliver a product that satisfies both the client and the people actually working with it.

What TU Delft expects from us

This section's purpose is to document additional agreements between the TU coach and us. For the regular project deadlines, please see the end of this section.

Weekly meetings with the TU coach will be held on mondays at 11.00 to discuss the past and next week, and the overall progress of the project. Documents (such as the meeting topics) that will be discussed will be emailed to the TU coach at its latest the evening before the day of the meeting.

Minutes will be made of every meeting, whether with the client, the TU coach or both. These minutes shall be sent to the TU coach as well.

Regular project deadlines:

- 19-04-2016: Project Plan
- 29-04-2016: Research Report
- 20-05-2016 - 27-05-2016: mid-project meeting (coordinator + team + client)
- 27-05-2016: SIG 1st submission
- 17-06-2016: Final Report
- 17-06-2016: BEP infosheet
- 17-06-2016: SIG 2nd submission
- 24-06-2016: Final presentation