Manifesto

Digital library team: BS19-02
Karina Singatullina,
Denis Schegletov,
Tasneem Toolba,
Danis Alukaev,
Aidar Khuzin.

Why is this important to you?

This Manifesto provides an overview of our primary goals, desired achievements, and main points of the development process. As a team member, you need to know the importance of processes and interactions within the team. I hope this document will help you get a better understanding of them.
Please note that this information may change from time to time.

Our focus.

The foremost significance of our business is assessed not by the course grade or the quality of the project, but by the team's coherence. Our core value is a system that we will create from a series of mistakes and incorrect decisions. Your future as a developer depends on how you learn the rules of teamwork right now. I try to set up the processes in our team in such a way that we remain as efficient as possible, but this requires **your** daily work and enthusiasm.

Our goals.

Primarily, you need to understand that life is a constant activity. You must learn to prioritize tasks in favor of the team. Keep in mind that your teammates depend on you. All work in Innopolis takes place at a serious pace, and in these conditions, you must have the skill to work productively and according to plan.
Besides, your ability to adapt to the situation and work in a team is important. You should know in advance what may be required of you and what is required of you now. Learn to understand each other, be united as childhood friends.
Two of the most powerful tools for our work are the git and the Scrum. In particular, all processes occurring in the git reflect on the Scrum board. It allows

us to manage the workload and decide objectively. Consequently, devote your time to learning these tools to be productive in the future.
And to achieve all these goals, we have a project. Apparently, this project should go through all development stages and be beautiful on both sides of the frontend and backend. You have a role, play it and make sure that at the end of our "performance" there will be applause.

Workflow.

Planning.

For each assignment, I create a plan in the form of a numbered list. Each entry contains a description of step, soft and hard redlines, the assignee and delivery format. I usually send a copy of this outline to the chat and ask you to confirm your acquaintance in any decent way.
The soft redline will always be 2 hours earlier and involves submitting your material for review. Further, the assignee will review your work and write comments. The hard redline means the delivery of the corrected material with all resolved issues.

Misconduct policy.

Penalized will be the following:
1. Failing the hard redline without legal excuse or warning in advance.
2. Absence at the meeting without legal excuse or warning in advance.

Git workflow and interaction with Scrum board:
0. The task card moved from **To-Do** to **Doing**.
1. Checkout *develop* branch: git checkout -b develop.
2. Create a new branch from *develop* using: git checkout -b <name>.

3. Write the code here.
3.1. In order to set the Git to track the changes of all newly created files: git add
3.2. To track only specific files: git add filename1 filename2 filename3 ... filenameN.
3.3. To exclude specific files or directories from Git tracking mention them in .gitignore file.

4. Commit the changes, i.e., create the code version checkpoint one will be able to return to in future:
git commit -m '+ <some_feature>'.
4.1. Push local changes to the remote repository.
The changes can include several commits: git push origin <name>.

5. Create Merge Request (MR) in branch *develop* (Assignee: Danis Alukaev).
6. Move the task card from **Doing** to **Review**.

7. Assignee checks the code, makes sure that the task was completed correctly.
7.1. If the implementation is nice, the assignee merges branch <name> into *develop* branch . Task card moves to **QA**, where the feature is tested.
7.2. If there is any issue in implementation, either conflicts with develop branch or non-optimal implementation or bad code style in implementation, then the task card moves to **Doing** state (all the issues listed in correspondent card).

8. If the implementation meets the requirements and passes all developed tests, the task card moves to **Done.** Otherwise, the Quality Assurance department undos commit, writes an explicit report and moves the card to **Doing**.

The cycle repeated for all features.
Release: after one sprint (2 weeks) the *develop* branch is merged into *master* branch.

Branches for developing of new functionality should be called:
feature/<description>
Branches for fixing previously developed functionality should be called:
fix/<description>
Commit message (git commit -m '<MESSAGE>').
1. You added new functionality: -m '+ <description>'.
2. You fixed the issue in previously developed code: -m '! <description>'.
3. You deleted useless staff: -m '- <description>'.
4. No changes in functionality, but the code became better or directory structure became more logical: -m '= <description>'.