Development protocol

```
Your code is your child, your legacy, it will live on after you are
gone.

Other people will need to read it, use it and develop it further.

If you leave a dirty legacy, people will be afraid to touch it and it
will die.

If you leave a solid legacy, people will build on it, and it will
flourish.
```

Section contents:

JIRA scenario

We use a combination of JIRA and Github for version control and code review.

Basic development scenario per JIRA swimlane is:

- "TODO"
  - ask all necessary questions about the functionality in comments, mention the people
  - create a confluence page in Product Description for large complex tasks and finalize the discussion there

    IMPORTANT NOTE
    Subdivide all JIRA tasks that take >1d into subtasks.

    If you cannot finish all subtasks in one sprint:

    1. convert the parent task into an **Epic**
    2. convert all subtasks to Tasks
    3. assign these Tasks to the Epic
    4. only leave the Tasks that you can accomplish within a sprint, move the rest to future sprint(s)
    5. for testing and code review:
    - open pull requests to merge Tasks' feature branches into Epic,
    - test Task branches in Stack-SOF, Epic branch inside Stack-Dev

  - update task description with acceptance criteria
- "IN PROGRESS"
  - create a separate branch corresponding to a JIRA task/issue
  - write tests
  - implement functionality
  - test locally on vagrant
  - adjust the code with necessary comments
- "TESTING"
  - create a build tag inside your feature branch
  - test remotely on ci (SOF-*), if build failed - you will receive an email
  - put a task to "Code Review"
- "CODE REVIEW"
  - create a pull request so reviewers can see all changes in one place (keep diffs below 1500 lines per on file)
  - include a link to successful SOF-* build into the last comment
  - include a link to any relevant Confluence pages into the last comment
  - wait for the reviewer-on-duty to "Accept" the code or put it back in "TODO"
- "MERGE
  - merge your feature branch into "dev" (dev-* build job will be automatically started)
  - if build failed, you will receive an email - come back and fix the build ASAP in this case as no one else will be able to merge their feature branches
- advance task to "DONE" when all builds passed

General guidelines

1. Before starting:
    a. Make sure there exists a JIRA issue that explains the reason for your development efforts
    b. Make sure that the issue is assigned to you
    c. Update the status of the issue to "In progress"
    d. Make sure there is only one issue with a status "In progress" at all times
2. Checkout `dev` branch for `exabyte-stack`

```
git checkout dev
```

3. Create branches with the name exactly equal to the issue tag (**feature/SOF-202** in the example below) for all repositories involved (replace the list of submodules with the one relevant for your case in the line below):

```
for i in web-app saltstack api lib/capy; do cd $i; git checkout -b
feature/SOF-202; cd -; done
```

    More information on the branching strategies for git is available here.

    For bug fix tasks that did not get to master yet, use `bugfix` as type, otherwise - use `hotfix`.

4. Apply your coding skills. Write tests. Don't forget to: commit often, perfect later, publish once. Basic guidelines for writing the code are available here.

    Remember that you must **keep diffs below 1500 lines** to make pull requests viewable on github. **`git diff --stat`** helps understand how many changes were introduced.

    Keep each pull request relatively small and have more of them instead. If diffs become larger than 1500 lines, split JIRA issue for it into 2 subtasks, create branches for them and inherit one from the other:

    - eg. if feature/SOF-333 has diff that doesn't fit into 1500 lines,
    - create feature/SOF-334, feature/SOF-335,
    - explain what was done in each in their descriptions and
    - open pull requests:
    -- from feature/SOF-333 to dev,
    -- from feature/SOF-335 to feature/SOF-334 and
    -- feature/SOF-334 to feature/SOF-333.

5. Test locally on vagrant. For example, for end-to-end tests run (@focus below is optional and is used to only run a subset of tests - the ones that you added, presumably)

```
./vagrant/vagrant-run-end-to-end-tests.sh @focus
```

6. Push your code to the remote repository on Github, **create a pull request for code review**. (Code review schedule is available on Devel oper Documentation)

```
git add -A .
git commit -m "This is what my magic did"
git push origin feature/SOF-202
```

For tasks containing subtasks: create a pull request from subtask into the main one.

For longer tasks, It is OK to create pull requests from one subtask to another to avoid them being stuck in `code review` stage.

7. Push to github to test your code on CI using Stack-SOF job (for feature branches). Advance JIRA task to "Code Review" after successful build and **include a link to the successful build**.
8. When the code is "Accepted" by one reviewer, merge the feature branches for the issue (SOF-202) into **dev** for `exabyte-stack` and all submodules involved

```
$ git checkout dev
$ git merge --no-ff feature/SOF-202
$ git branch -d feature/SOF-202
```

9. Acceptance tests are automatically triggered after a push to **dev**. In case of build failure - fix the **dev** branch immediately and comment on the reason why tests succeeded in Stack-SOF but failed in Stack-Dev inside JIRA task.
10. On success, transition the issue to Done in JIRA and **include a link to the successful build**.

Links

1. https://datasift.github.io/gitflow/IntroducingGitFlow.html