Enabling researchers to develop and deliver software in an agile way – The rate of research in bioinformatics is high, and researchers often do not have time to set up and carry out a business-level software development process. However, one would still like to be able to quickly write, bug-fix and deploy functionality to users (including oneself), so that one can always deliver the best solution to the problem at hand.

Galaxy ProTo

We have implemented the present development model in the Genomic HyperBrowser team (https://hyperbrowser.uio.no) at the Department of Informatics, University of Oslo. Even though the concepts can be adapted also for traditional Galaxy development, we have benefitted greatly from Galaxy ProTo (https://github.com/elixir-no-nels/prot0), which is a tool building methodology introduced by us as an unofficial alternative for defining Galaxy tools. Galaxy ProTo provides rapid and simple development of Galaxy tools using Python code only, skipping the need for writing a command-line argument parser and a Galaxy XML wrapper. Galaxy ProTo enables a fully dynamic user interface and on-the-fly development. In practice, the developer can develop a web interface in a Python integrated development environment (IDE) and then just switch to a web browser to see the changes in real-time, with no need to use the command line and little need for other tools. Note, however, that Galaxy ProTo tools are incompatible with Galaxy workflows.

Galaxy for development, testing and production

Galaxy is used in all parts of the development process. When one is fixing bugs, it is very powerful to be able to redo a failed analysis by a single click, especially if some time has passed between bug detection and fixing, or if another user found the bug. Having automatic storage of previous bugs in a history is also sometimes handy. Especially useful is the bug report feature of Galaxy, which we use to automatically create Trello (https://trello.com) cards to track the requests.

Continuous delivery onto a Galaxy server

Central to our development process is the idea that the researcher uses his/her laptop for development only (using an IDE), but runs all code on from Galaxy instances on a shared server. This keeps the developer from having to understand Galaxy internals in order to deploy a test instance locally. More importantly, it enables the researchers to carry out both test and real jobs that spends large resources without having to tax a limited laptop (which one can then close and move without interfering with running jobs). We use two alternative delivery systems to carry out the syncing of code to the server:

- Synching (https://synching.net) is used to move any code changes in real time from the laptop to one of two personal Galaxy instances on the server, one for development and one for running jobs ("sandbox").
- GoCD (https://www.gocd.org) is used for shared project servers that host suites of related tools. GoCD automatically builds, tests and sets up a clean Galaxy server whenever a new commit is pushed to a dedicated branch on a GitHub repo. Several developers can, in this way, easily work on the same project and move code into testing/production when needed. At the time of paper submission, one just cleans up the project instance a bit and can use it to host live examples, results and publicly available functionality that will remain unchanged for reproducibility purposes.

Cutting corners

We acknowledge that research is a domain where efficiency is important, and also that many researchers are not well versed in best practices in computer science. We thus allow (and promote) cutting of unit tests and other time-consuming endeavors (e.g., documentation) if one is prototyping features or if one decides that supporting boundary cases correctly might not be that important. The responsibility is with the researcher. However, we still advocate for the advantages of unit testing (and specifically test-driven development) for robustness, clean coding, and regression testing, if one has time for it. This recommendation is based upon experience with using our process (and previous iterations) in a total of 21 publications on novel methodology and applications.