Closing the feedback loop: Community driven development of a file format

Adrian Stoewer [1], Christian Kellner [1], Jan Benda [2], Michael Sonntag [1]
Andrey Sobolev [1], Thomas Wachtler [1], and Jan Grewe [1,2]

[1]German Neuroinformatics Node, Department Biologie II,
Ludwig-Maximilians-Universität München, Germany;
[2]Institut für Neurobiologie, Universität Tübingen, Germany

Introduction

Looking back on the development process of the NIX[1] project we would like to share our experiences with the community, seeking fruitful discussions and exchange of information about community-driven software development in a scientific working environment. We encountered challenges and issues at technical, logistic, and sociological levels. One aspect, which is probably common to many such projects, is that resources are limited, and most of the development relies on a small group of contributors with diverse backgrounds, located at different places. Thus, it is important to improve communication, collaboration, and code quality management. Second, while it is crucial that the development is guided by multiple use cases, scientists, even those that are interested in using the software, are typically busy with their everyday research. This makes it difficult to obtain information, data and feedback, especially in the initial phases when the funcitonality of the software is still limited. Furthermore, in contrast to software that is written for use within a single lab, developing software that is intended to be useful for a broader community poses substantial demands.

Challenges and solutions in community driven software development

Challenges:
- Analyze the problem and identify an appropriate set of requirements.
- Plan features of the format according to realistic requirements

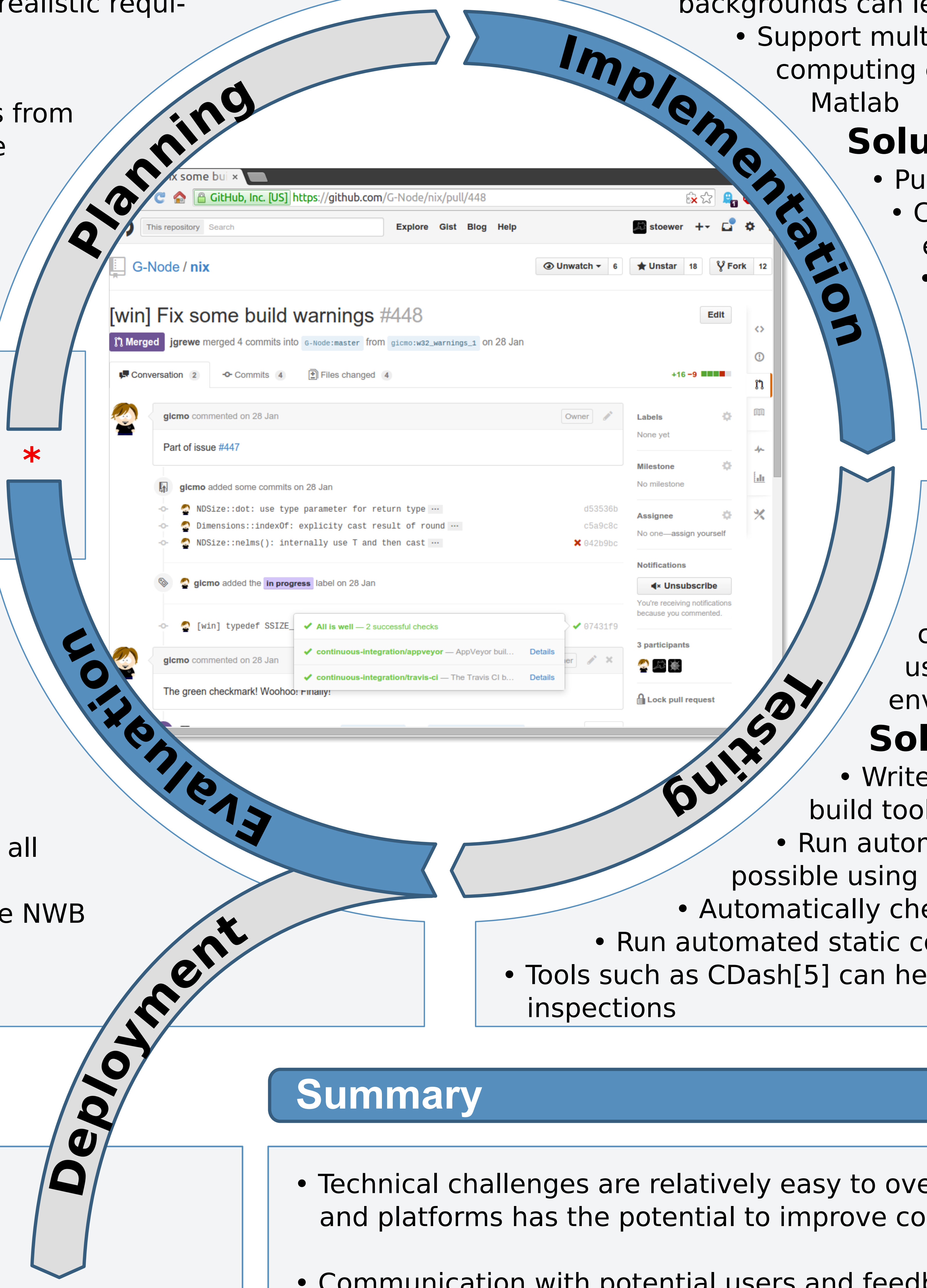
Solutions:
- Participate in and interact with initiatives from the neuroscience community such as the Electrophysiology Task Force of the INCF
- Have scientists and experimentalists on the development team

Feedback Gap

Chicken-and-egg situation: real feedback can only be obtained from the final format which by itself needs feedback during its development *

Challenges:
- Get feedback from the community in order to evaluate implemented features
- Test implementation against real world use-cases

Solutions:
- Collaborate with labs, thus ensuring that all requirements are met
- Get use-cases from initiatives such as the NWB project



https://github.com/G-Node/nix/pull/448

G-Node / nix

[win] Fix some build warnings #448

Merged jgrewe merged 4 commits into `G-Node:master` from `gicmo:w32_warnings_1` on 28 Jan

gicmo commented on 28 Jan

Part of issue #447

gicmo added some commits on 28 Jan

```
NDSize::dot: use type parameter for return type …          d53536b
Dimensions::indexOf: explicity cast result of round …       c5a9c8c
NDSize::nelms(): internally use T and then cast …        ✗ 042b9bc
```

gicmo added the in progress label on 28 Jan

```
[win] typedef SSIZE_                                     ✓ 07431f9
```

✓ All is well — 2 successful checks
✓ continuous-integration/appveyor — AppVeyor buil…
✓ continuous-integration/travis-ci — The Travis CI b…

gicmo commented on 28 Jan

The green checkmark! Woohoo! Finally!

Labels: None yet
Milestone: No milestone
Assignee: No one—assign yourself
You're receiving notifications because you commented.

Challenges:
- Heterogeneous teams consisting of members from different backgrounds can lead to varying code quality
- Support multiple programming languages and computing environments such as Python and Matlab

Solutions:
- Pull-request based development cycle
- Code review and four-eyes principle on each pull-request
- Implement library in C or C++ to support future language bindings -> slows down the development

Challenges:
- Ensure code quality and correctness
- Achieve cross-platform and cross-compiler support although developers usually work in their preferred environment

Solutions:
- Write unit tests and integrate them in the build tool-chain
- Run automated tests on as many platforms as possible using services like Travis-CI[2] or Appveyor[3]
- Automatically check test coverage with Coveralls[4]
- Run automated static code analysis
- Tools such as CDash[5] can help to get an overview about all inspections

Challenges:
- Create installers and packages for all platforms as expected by their users

Solutions:
- Use standard build tools such as cmake, setuptools, and debuild
- Build services can help to automatize the process
- Use distribution platforms such as PyPi, Maven Central, Homebrew and Launchpad

Summary

- Technical challenges are relatively easy to overcome. A rich ecosystem of services and platforms has the potential to improve colaborative development greatly.
- Communication with potential users and feedback from the scientific community is still an issue.
- Currently there is only limited incentive for scientists to perform non scientific work such as contributing code to software projects or testing new software during the development.

*Can organizations like the **INCF** bridge the gap between developers and potential users and give scientists more incentive to contribute to software projects?*

Resources

[1] https://github.com/G-Node/nix
[2] https://travis-ci.org
[3] http://www.appveyor.com
[4] https://coveralls.io
[5] https://open.cdash.org

contact: adrian.stoewer@rz.ifi.lmu.de

This work was performed in connection with the activities of the HDF5 working group of the INCF Electrophysiology Data Sharing Task Force

Supported by:
G-Node (BMBF grant 01GQ1302)

Federal Ministry
of Education
and Research