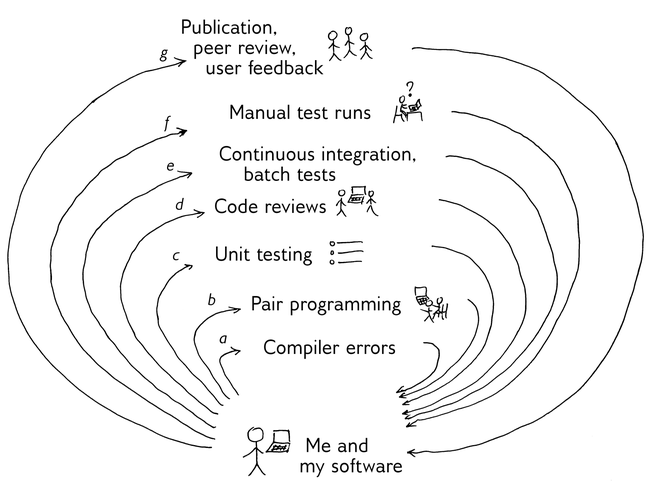
SOFTWARE TECHNIQUES FOR GOOD PRACTICE IN AUDIO AND MUSIC RESEARCH

Luís A. Figueira, Chris Cannam and Mark D. Plumbley
Queen Mary University of London

Paper 8872

In this paper we discuss how software development can be improved in the audio and music research community by implementing tighter and more effective development feedback loops. We suggest first that researchers in an academic environment can benefit from the straightforward application of peer code review, even for ad-hoc research software; and second, that researchers should adopt automated software unit testing from the start of research projects. We discuss and illustrate how to adopt both code reviews and unit testing in a research environment. Finally, we observe that the use of a software version control system provides support for the foundations of both code reviews and automated unit tests. We therefore also propose that researchers should use version control with all their projects from the earliest stage.

FEEDBACK CYCLES IN SOFTWARE

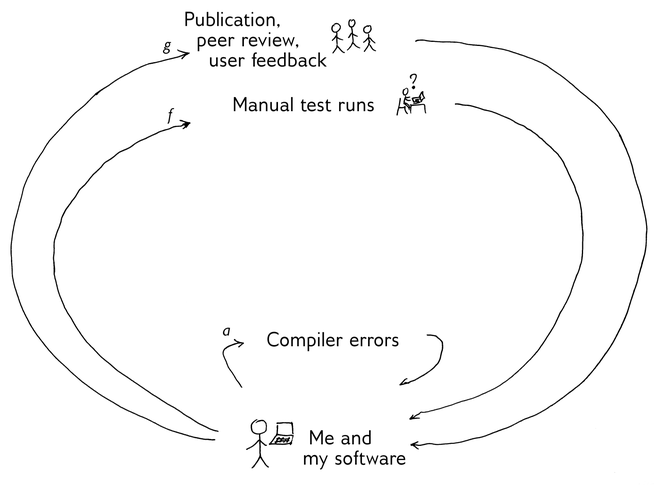


The importance of feedback
• allows developers to learn about software mistakes

Commercial best practice
• short/simple cycles
• find mistakes quickly

Frequently used techniques
• pair programming, continuous integration, unit testing

TYPICAL RESEARCH WORKFLOWS



Compiler/IDE feedback
• no information on program correctness

Manual test runs
• difficult to distinguish genuine results from bugs

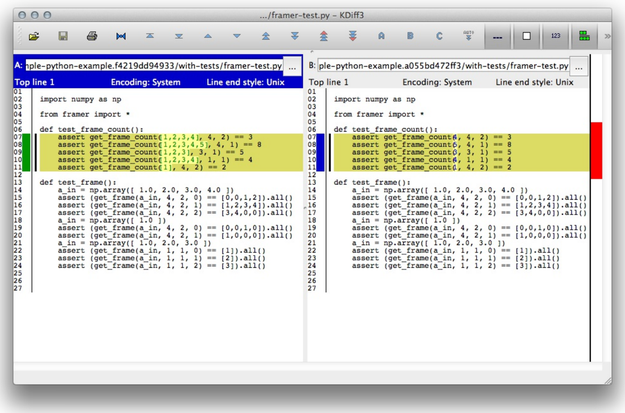
Publication/User feedback
• can take a long time
• requires code/data publication

CODE REVIEWS

More effective at finding errors than any other commonly used technique [Fagan, 1976]

Can be carried out quickly and informally in a peer setting such as a research lab [Dunsmore et al, 2000]

Reading code and writing readable code are valuable in developing software development ability [Deimel et al, 1990]

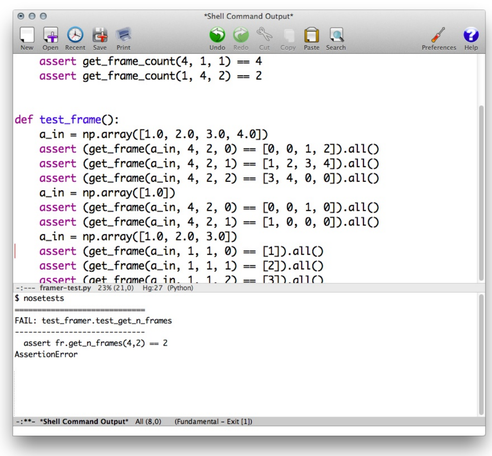


UNIT TESTING

Relatively easy way to obtain any assurance of correctness

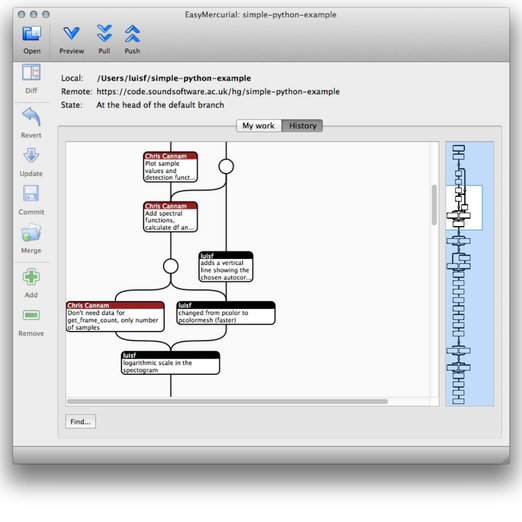
Good way to defend against regressions, which are hard to discover simply by manual testing

Test driven development (TDD) can provide an additional analytical perspective when solving difficult problems [Erdogmus et al, 2005]



VERSION CONTROL

General management of code becomes easier



Eases code sharing between developers, such that they can be sure they are looking at the same version of the same software

Provides the necessary support to compare two versions of code and identify the source of a change in behaviour

RESEARCH GROUP RECOMMENDATIONS

Apply an informal style of "use-case" code review technique
• peers should review the code before important experiments
• use "pre-merge" code reviews when working with shared repositories

Adopt the simplest available unit testing regime
• keep unit tests small and concise
• use a standard test framework for unit tests on larger programs

Use a version control system
• should be used from the beginning of any software project
• fundamental to support both code reviews and unit testing

Read our paper!

soundsoftware.ac.uk

Complete article, bibliographic references and further information available at:

http://soundsoftware.ac.uk/aes-rome2013-paper

EPSRC
Pioneering research and skills

CC BY SA