SOUND SOFTWARE: TOWARDS SOFTWARE REUSE IN AUDIO AND MUSIC RESEARCH

Chris Cannam, Luís A. Figueira and Mark D. Plumbley

Queen Mary University of London
Centre for Digital Music
{chris.cannam, luis.figueira, mark.plumbley}@eecs.qmul.ac.uk

ABSTRACT

Although researchers are increasingly aware of the need to publish and maintain software code alongside their results, practical barriers prevent this from happening in many cases. We examine these barriers, propose an incremental approach to overcoming some of them, and describe the Sound Software project, an effort to support software development practice in the UK audio and music research community. Finally we make some recommendations for research groups seeking to improve their own researchers' software practice.

***Index Terms*—** Programming, Reproducible research, Software reuse, Software tools, Scientific computing

1. INTRODUCTION

Much research in audio and music informatics involves the development of new computational methods implemented in software and the evaluation of new methods against earlier work also implemented in software. Both of these can be problematic in practice.

First, researchers in the audio and music research community—including those in the group represented by the present authors, the Centre for Digital Music (C4DM) at Queen Mary University of London—come from a wide range of backgrounds besides signal processing, including electronics, computer science, music, information sciences, dance, performance, and data sonification. In many of these fields, researchers do not have the skills or desire to become involved in traditional software development practice or in publication and maintenance of code.

Second, there are technical and logistical reasons why software developed during earlier research becomes unavailable for subsequent use or development even if it has been published. These include platform incompatibilities and obsolescence, or legal limitations on distribution or reuse.

In this paper we discuss some of the practical constraints on application of reproducible research principles in connection with reuse of research software. We then explore an incremental approach toward better practice. Finally we make some early recommendations for research groups that wish to improve software development practice in their work.

2. REPRODUCIBLE RESEARCH

Some researchers have come to realize that traditional methods of disseminating research outputs based on the published paper only are no longer sufficient for computational science research. The algorithms and parameters involved are often so complex that the description in the paper is no longer sufficient to reproduce the results. As an alternative approach, Donoho and colleagues at Stanford have, since the mid 1990s, aimed to carry out "Reproducible Research" by providing the paper, source code, and data, sufficient for other researchers to reproduce the same results [1].

Recent years have seen some moves to promote this philosophy across the signal processing research community. A special session was organised at ICASSP 2007 [2], and special issues of IEEE Signal Processing Magazine [3] and Computing in Science and Engineering [4] on this subject both appeared in 2009. The IEEE Signal Processing society now encourages Reproducible Research, allowing links from the online journal repository IEEE Xplore to the code and data associated with a publication [3].

Actions such as these promote the idea that research results in signal processing should be presented not simply as a printed paper, but as a *compendium* [5] including the paper, research data, and code. Vandewalle et al [3] also created a Reproducible Research Repository[1], designed to promote reproducible research by requiring the authors of a paper to upload the code and data used in the experiments. Readers can then comment on a publication and evaluate the reproducibility of the work.

However, although the Reproducible Research principle proposes a comprehensive solution to the problem of code dissemination, our experience has been that take-up in the audio and music research field is limited. Why?

3. UNDERSTANDING REAL-WORLD LIMITATIONS ON SOFTWARE PRACTICE

In order to better understand the reality faced by the audio and music research community, we conducted an online survey on software usage and development [6]. This survey opened in October 2010 and was advertised to a number of senior researchers in other groups around the UK. We asked for detailed information about the software usage, authorship and publication practices of researchers, with the aim of obtaining a number of individual case points for further examination as well as some broad numerical results. The survey closed in April 2011, with 54 complete and 23 partially complete responses. There were responses from at least 16 different institutions, with a number of common issues reported.

Some 80% of respondents reported developing software themselves during research and 40% of those said that they took steps to ensure reproducibility of their publications. Respondents described taking a number of positive steps such as using standard, publicly-available datasets and calibration procedures when performing measurements, or documenting code and data so that they or other researchers in their group could reproduce the results later.

However, respondents' comments suggested that they did not always count publication of software among their steps toward re-

[1] `http://rr.epfl.ch/`

producibility. Only 35% of those respondents who reported both developing software and taking steps to reproducibility also reported having in fact published any code.

Our respondents cited as obstacles to the publication of code lack of time, copyright restrictions, and the potential for future commercial use. A broader study into science research across several subject areas by the UK Research Information Network [7] additionally identified inhibiting factors for open sharing of data and code including: lack of evidence of benefits, cultures of independence and competition, and quality concerns. Similar findings appear in [8].

Besides these practical obstacles, undertaking reproducible research takes effort early in the research cycle. This happens before the benefits are necessarily apparent, and while the value of the research is still unclear. Once results have been produced and a paper written there is little apparent incentive to make the research reproducible. Furthermore, assessments such as the English "Research Excellence Framework" [9] typically do not identify software code among assessed research outputs.

A possible reason why reproducibility efforts do not happen earlier is that researchers are often self-trained in software development. Therefore they make little use of standard software engineering practices that could facilitate open dissemination of code, such as collaborative development and the use of public code repositories. A study by Hannay et al [10] found that for developing and using scientific software, informal self-study or learning from peers was commonplace. The same study found that scientists usually developed and used software on their own desktop computers, rather than servers provided for the purpose of running scientific software. In our survey, 51% of respondents who developed software said that their code did not leave their own computer, and 59% said they did not use any version control software.

Not only does software often go un-published; software that is published is often unavailable for future users because of platform incompatibilities. For example, in the well-known subject of musical beat tracking, the method of Scheirer et al [11] was written for a legacy platform and is now only available by informal means; that of Goto [12] was written for a parallel architecture no longer in wide use and never publicly released; and that of Hainsworth [13] was written in MATLAB with a Windows-specific DLL component and only runs on a single platform. In many of the fields within this community, researchers lack the skills or desire to grapple with code if it will not immediately run on a platform they have available. Where they do produce code, they use a variety of platforms and batch and real-time environments. Among technologies used by respondents to our survey were MATLAB and numerous of its toolboxes, Max/MSP, C++ and OpenFrameworks, Juce, HTK and MPTK, SuperCollider, Python, and Clojure.

4. SUSTAINABLE SOFTWARE: A BOTTOM-UP APPROACH

Our approach to the sustainable software problem is to facilitate incremental improvements to the way software is managed during research. We have attempted to do this by identifying those practical barriers to software reuse that admit straightforward removal or mitigation through simple educational and technical measures.

While we support the goal of reproducible research, and aim to encourage open publication of code and data linked with paper publications, we believe that this goal is more easily approached *incrementally*. We maintain that researchers will appreciate improvements to software development practice, regardless of their beliefs or intentions with regard to reproducible research. By helping researchers to feel comfortable with managing provenance and versioning for software, with collaborative development of code, and with the perception of code as something that may readily be reused, we aim to prepare ground in which open and reproducible publication can naturally grow.

Further, the software lifecycle does not end with publication. Software that is to be used needs maintenance, and any proposal to help researchers reuse software more easily needs to address the problem that such software is not always in a reusable state. Our direct concern therefore is *sustainability* and *reusability* rather than *reproducibility*.

While we cannot address all possible barriers to software publication and reuse, from the issues arising in Section 3 above we identify four specific barriers that we consider to be approachable: lack of education and confidence with code; lack of facilities and tools to support collaborative development; lack of incentive to distribute software (given the academic focus on paper publications); and reusability problems caused by platform incompatibilities.

4.1. Barrier: Lack of education and confidence with code

In Section 3 we noted that researchers are largely self-trained in software development. This can lead to problems with software structure and lack of testing [14]. Although software development is a deep subject, our belief is that worthwhile improvements to normal working practice can follow relatively small amounts of training.

Therefore in November 2010 we organised an Autumn School for researchers, presented by Dr Greg Wilson and based on his Software Carpentry materials [15]. This week-long residential course, for 20 audio and music researchers from groups around the UK, taught fundamentals of software development. These included version control for software, unit testing and test-driven development, Python syntax and structure, and managing experimental datasets with sqlite. We have made available all of the teaching material from the Autumn School in online videos.[2]

A subsequent online poll of attendees [16] supported the view that training in even the most basic software development skills may be well received by, and beneficial to, researchers. Attendees identified program design, testing and validation, and provenance and reproducibility as particularly valuable areas covered. These are areas in which the simplest possible introductions to program structure, test-driven development, and version control can provide sufficient provocation for the researcher to re-evaluate their own practices.

4.2. Barrier: Lack of facilities and tools

4.2.1. Facilities for code hosting and version control

Researchers cannot make use of version control and collaborative development facilities if they are unavailable or unknown to them. Few of the attendees at our Autumn School (Section 4.1) were aware of such facilities being provided by their institutions, and in our survey (Section 3) only 41% of respondents who wrote software said they ever used them. This is consistent with experience in our own group, where version control has been used only sporadically.

To address this issue, we developed the SoundSoftware code site[3] as a service which audio and music researchers in the UK may use for collaborative development and as a version control and code hosting facility. The site is designed to help researchers whose institutions have no suitable facility or who need to collaborate with

[2] http://soundsoftware.ac.uk/autumnschool2010video/
[3] http://code.soundsoftware.ac.uk/

individuals at other institutions in a way that their own facilities do not support. Any UK researcher in the field can register and start their own projects. Research groups as a whole may also make use of the site, and at C4DM we use it to provide version control to our own researchers.

The site is implemented using a custom version of the Redmine[4] project management application, together with Mercurial distributed version control. The software implementation of our code site is public, available through a project on the site itself.

We designed three aspects of the code site to contribute to sustainability and code reuse for researchers, distinguishing this site from general-purpose code hosting facilities such as SourceForge[5], Google Code[6] or GitHub[7]:

1. *Focus* — The focus of the site on audio and music research is intended to make researchers who do not think of themselves as software developers feel that they are among peers, and to make it easier to locate and obtain relevant code.
2. *Public and private projects* — Projects can be entirely public, or private to a group of collaborating researchers; work can also be started privately and made public later. We believe that supporting private projects helps users become comfortable with the site. At the time of writing 57% of projects hosted at the site are private, and even for private projects the average number of members is almost 2.
3. *Linking publications with code* — Users can associate publication records with their projects, so that readers can immediately see what publications are related to the code (see Section 4.3).

The site is also capable of tracking external projects. Researchers who use code hosting or project management facilities elsewhere can also make use of our site as a nexus for relevant projects, registering their projects at our site and making it point to their own hosting.

We believe these features together will encourage researchers to employ collaborative development early in their work, and to place themselves in a situation in which the outcomes of their work can be used in a sustainable way with relatively little extra effort.

4.2.2. User interfaces for version control

Attendees at our Autumn School also reported difficulty during the course in getting started with the complex user interfaces available for version control. Nonetheless, version control was amongst the areas identified subsequently as most valuable.

To address some of the difficulties faced in learning version control we have developed EasyMercurial,[8] an application designed to be easy to teach to researchers across multiple operating system platforms. This application uses a visual graph representation for change history, showing explicitly the flow of simultaneous changes by different users and of merge points. EasyMercurial behaves identically on each supported platform (Windows, Mac OS/X and Linux) in order to simplify training for researchers with disparate platforms.

[4] http://redmine.org/
[5] http://sourceforge.net/
[6] http://code.google.com/
[7] http://github.com/
[8] http://easyhg.org/

4.3. Barrier: Lack of incentive for publication

In Section 3 we noted that software and data are typically not recognised as citeable or assessable research outputs. Software also lacks the publication convention for describing the authorship hierarchy, making it unclear how an academic should be recognised for their contribution to a collaborative software work.

Our code site permits users to give publication references for their code, which are shown on their project's front page. This not only increases the likelihood of the code being discovered by users searching for publications by title, but also ensures that anyone seeking the code will know how to cite its methods in their own publications, increasing the citation impact of the work.

4.4. Barrier: Platform incompatibilities

We observed in Section 3 that researchers in this field choose to use many platforms and programming languages to carry out their work. Although the most common (MATLAB) is used in many signal processing groups, it is a commercial platform that is not widely used in other fields related to audio and music, such as computational musicology or music therapy.

To promote software reuse outside of the immediate signal processing community, C4DM has adopted a "plugin" approach. Writing a plugin allows a working algorithm to be converted directly to a unit of code which can be used in real applications, without the need to develop a custom user interface. Developing to a published specification supported by more than one host program increases the relevance of the code and therefore the likelihood of its being maintained. Code that uses the plugin format of a successful application is relatively likely to be understood by other developers.

At C4DM we have had success with plugins in standard audio processing formats such as VST[9] as well as in writing externals for modular systems such as Max/MSP or SuperCollider. For audio analysis methods, in 2006 we developed the Vamp plugin system [17] and implemented it using C++ in our visualisation software Sonic Visualiser [18], a widely-used application that has seen over 200,000 downloads to date, and in our Sonic Annotator batch feature extractor. The Vamp system has subsequently been used by C4DM and others with some success for publishing working methods.

5. CASE STUDY: CHORDINO AUTOMATIC CHORD TRANSCRIPTION

Mauch [19] describes a method for improving automatic recognition of chords by a prior approximate note transcription step. This is a traditional publication which appeared without accompanying code or test data. Although no formal attempt was made initially toward reproducibility, some independent evaluation was carried out through the submission of a MATLAB implementation of the method to the annual MIREX evaluation exchange [20].

Following this publication, the author worked with us to develop a C++ implementation of the method and turn it into a Vamp plugin for chord estimation, named Chordino. This code and its revision history are available through our code site[10] and thereby linked with the associated publication. Although the code has been updated since release, as a reproducibility aid the plugin includes a mode in which it uses the same method as that submitted to the MIREX evaluation.

[9] http://en.wikipedia.org/wiki/Virtual_Studio_Technology
[10] http://code.soundsoftware.ac.uk/projects/nnls-chroma

As a consequence, even though this process did not begin until after the initial publication, a high degree of openness and effective reusability have been achieved.

6. RECOMMENDATIONS

Research groups wishing to improve software development practice for their researchers have a number of options which may prove relatively straightforward to carry out.

Aim at easy training targets. In Section 4.1 we observe that training which may be considered simplistic in the context of software engineering, such as an introduction to program structure across multiple source files and functions, may yield tangible rewards to researchers who are technically minded but lack software development experience and fear writing code. We encourage researchers to make use of the video material from our Autumn School[11] and the related Software Carpentry course material.[12]

Provide version control software and hosting and encourage researchers to use it. Version control is also useful when writing papers in formats such as LaTeX, as well as for code. If you are in the UK, consider using the code site we offer (Section 4.2.1).

The benefits of version control—managing software history and enabling collaborative development—are sufficiently abstract that it may be more effective to show than to explain them. Our EasyMercurial application (Section 4.2.2) is designed to be an effective teaching tool for visually based tutorial sessions.

Turn code into plugins and seek other ways in which your code can be used in conjunction with end-user applications that are already popular (Section 4.4).

Encourage collaborative development. Researchers are routinely encouraged to work together on conference papers, but collaborative software development appears to be the exception rather than the rule (Section 3). Working together increases the opportunities for learning and creates an environment of confidence about sharing and reusing code.

7. CONCLUSIONS AND FUTURE WORK

We have identified incremental changes to software development practice for audio and music researchers, and described the work we have done in support of them.

We found that many researchers lacked confidence with code, which we have begun to address with basic software development training. We found many researchers did not use code management and collaborative working facilities, so we provided a code hosting site with a focus on audio and music research software, and easy version control tools to use with it. We found that published code may not always be academically recognised or cited, so we have strengthened the connection between software and citations in our code site. Finally we found that platform incompatibilities prevent software being reused, so we propose providing code in the form of plugins for widely used applications where possible.

In future work, we will need to evaluate the results of these changes in terms of the proportion of publications with published and reusable code. We are planning to follow up the 2010 Autumn School for researchers, possibly using a localised model to take the school to a number of locations around the UK. And we plan to produce more training material for version control and our code site, and evaluate how researchers respond to training with these facilities.

[11] http://soundsoftware.ac.uk/autumnschool2010video

[12] http://software-carpentry.org/

8. REFERENCES

[1] J. B. Buckheit and D. L. Donoho, "Wavelab and reproducible research," Tech. Rep., Stanford, 1995.

[2] J. Kovacevic, "How to encourage and publish reproducible research," in *Proc. ICASSP*, 2007, vol. 4, pp. 1273–1276.

[3] P. Vandewalle, J. Kovacevic, and M. Vetterli, "Reproducible research in signal processing—what, why, and how," *IEEE Sig. Proc. Mag.*, vol. 26, no. 3, pp. 37–47, 2009.

[4] S. Fomel and J. F. Claerbout, "Guest Editors' Introduction: Reproducible Research," *Computing in Sci. Eng.*, vol. 11, no. 1, pp. 5–7, 2009.

[5] R. Gentleman and D. Temple Lang, "Statistical analyses and reproducible research," *Journal of Computational and Graphical Statistics*, vol. 16, no. 1, pp. 1–23, 2007.

[6] I. Damnjanovic, L. A. Figueira, C. Cannam, and M. D. Plumbley, "SoundSoftware.ac.uk Survey Report," http://code.soundsoftware.ac.uk/documents/17, 2011.

[7] RIN, "Open to All? Case studies of openness in research," http://www.rin.ac.uk/our-work/data-management-and-curation/open-science-case-studies, September 2010.

[8] N. Barnes, "Publish your computer code: it is good enough," *Nature*, vol. 467, no. 7317, pp. 753, 2010.

[9] HEFCE, "Assessment framework and guidance on submissions," http://www.hefce.ac.uk/research/ref/pubs/2011/02_11/, July 2011.

[10] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How do scientists develop and use scientific software?," in *Proc. SECSE*, 2009.

[11] E. D. Scheirer, "Tempo and beat analysis of acoustic musical signals.," *JASA*, vol. 103, no. 1, pp. 588–601, 1998.

[12] M. Goto, "An audio-based real-time beat tracking system for music with or without drum-sounds," *JNMR*, vol. 30, no. 2, pp. 159–171, 2001.

[13] S. Hainsworth, "Beat tracking and musical metre analysis," in *Sig. Proc. Methods for Music Transcription*, A. Klapuri and M. Davy, Eds., vol. 4, chapter 4, pp. 101–129. Springer, 2006.

[14] Z. Merali, "Computational science: ...Error," *Nature*, vol. 467, no. 7317, pp. 775–777, Oct. 2010.

[15] G. Wilson, "Software Carpentry: Getting scientists to write better code by making them more productive," *Computing in Sci. Eng.*, vol. 8, no. 6, pp. 66–69, 2006.

[16] L. A. Figueira, C. Cannam, and M. D. Plumbley, "Autumn School for Audio and Music Researchers Survey Report," http://code.soundsoftware.ac.uk/documents/19, 2011.

[17] C. Cannam, "The Vamp Audio Analysis Plugin API: A Programmer's Guide," http://vamp-plugins.org/guide.pdf, 2007.

[18] C. Cannam, C. Landone, M. B. Sandler, and J. P. Bello, "The sonic visualiser: A visualisation platform for semantic descriptors from musical signals," in *Proc. ISMIR*, 2006, pp. 324–327.

[19] M. Mauch and S. Dixon, "Approximate note transcription for the improved identification of difficult chords," in *Proc. ISMIR*, Utrecht, Netherlands, 2010, pp. 135–140.

[20] J. S. Downie, "The music information retrieval evaluation exchange (2005–2007): A window into music information retrieval research," *AST*, vol. 29, no. 4, pp. 247–255, 2008.

This work was supported by EPSRC Grant EP/H043101/1. Mark D. Plumbley is also supported by EPSRC Leadership Fellowship EP/G007144/1.