# Communication and Conflict Issues in Collaborative Software Research Projects

Cornelia Boldyreff, David Nutter and Stephen Rank
*Faculty Of Applied Computing Sciences*
*University of Lincoln*
*{cboldyreff,dnutter,srank}@lincoln.ac.uk*

## Abstract

*The Open Source Component Artefact Repository (OSCAR) was developed under the auspices of the GENESIS project to store data produced during the software development process. Significant problems were encountered during the course of the project in both the development itself and management of the project. The reasons for and potential solutions to these problems are examined with the intention of developing a set of guidelines to enable participants in other collaborative projects to avoid these pitfalls.*

*We wish to make it clear that we attach no opprobrium to any of the participants in the GENESIS project as many of the issues we outline below have solutions only visible with hindsight. Instead, we seek to provide a fair-minded critique of our role and the mistakes we made in a fairly typical two-year EU research project, and to provide a set of recommendations for other similar projects, in order that they can (attempt to) avoid suffering similarly.*

## 1. Introduction

The GENESIS [7] platform is an Open Source software engineering environment designed to non-invasively complement an organisation's existing development practices. The GENESIS project initially developed all of the components that comprise the platform as a closed-source research project, however the intention from the start was to release the software as Open Source [3]. One component, developed at the University of Durham and now maintained and evolved in new projects at the University of Lincoln, is OSCAR [10, 4, 5]. This tool is intended to support the storage and retrieval of large collections of heterogeneous software artefacts. The other components of GENESIS include a work-flow management system and a project management tool, including a metrics generation and browsing tool. The consortium itself consisted of four development (academic) partners and two industrial partners.

Numerous problems occurred in each phase of the project, some of which affected the success of the OSCAR tool within GENESIS and some which have impacted upon the chances of successfully using OSCAR after the end of the GENESIS project. These problems were either common in other research projects or caused by circumstances peculiar to the GENESIS project.

Problems that are also found in other software development research projects include the effects of short timescales and rapid staff turnover. Even simple mistakes may not be rectified in time to prevent adverse effects later on in the project and the loss of access to key developers, often PhD students, fixed-term contract researchers, or students employed for short-term summer projects, and the ensuing loss of their knowledge may hamper ongoing maintenance of the software produced. Multiple re-writes of existing functionality--rather than steady evolution--are also common in research projects. Certain pieces of academic software have been re-written several times instead of being evolved from their original source code. An example of this approach is the development of the CodeWalker visualisation tool [9]. Often this approach is used because the original software cannot easily be adapted to new research needs, or the program comprehension overhead precludes re-use. Re-writes may occur for other reasons as well; for example the replacement of the Berkeley Packet Filter in OpenBSD with an alternative with a less restrictive license.

The development practices within research projects are often chaotic. For the most part, this is not a problem locally as development teams are small and thus communication overheads and conflicts within teams are minimized. When collaboration between multiple partners is required, an agreed process, even if unwritten, is necessary for effective collaboration.

## 2. Issues Within The GENESIS Project

A major flaw was over-ambition; the research goals of OSCAR were both lofty and difficult to achieve and evaluate successfully in a short timescale. The mere presence of lofty goals is not in itself a problem since they do ensurethat there are possibilities for future research and that these possibilities are under consideration in the earliest phases of the project. However, researchers as software developers should be very careful about the goals they seek to implement in the

context of their immediate project. Of course when applying for research funds, proposals must convince their reviewers that the proposed research will tackle difficult problems and explore innovative solutions, possibly involving speculative elements.

OSCAR also aimed to solve numerous problems experienced by our industrial partners, instead of a single problem at first and subsequent attempts to solve others. Consequently the initial requirements were very broad and complex. A concerted effort to decompose these requirements into simpler, smaller sub-requirements would have made the chosen rapid-release approach feasible. As it was, to satisfy even one requirement necessitated a significant amount of work on infrastructure, design and development before the code to address the requirement could be written. The requirements placed upon OSCAR by the other components in the system meant that significant functionality was required before OSCAR could be integrated with those components.

Finally, all parts of the GENESIS projects failed to provide suitable prototypes or mock-ups at the earliest possible phase of the project meaning the industrial partners were often unclear about what each platform component was supposed to do or how they would interact with it. Simpler individual requirements within OSCAR would have made implementing a very, very basic prototype early on in the project much easier.

While most of the features were desirable if OSCAR was to be a proper production tool, they were not necessary for a proof-of-concept and should have been relegated to later phases of the project. Following the principle of "release early, release often. . . " [11], a better approach would have been to drastically simplify the core of the initial version of OSCAR, aiming to have a very simple (even simplistic) version available as soon as possible, enabling the OSCAR team to fulfil the second half of that dictum: ". . . And listen to your customers". Good advice for all development teams is to play the expectations game; promise very little and over deliver. Doing the converse and thus disappointing one's consortium partners causes tension. Satisfying the project's reviewers may be in conflict with this strategy however, if it is not explicitly clarified.

One significant problem peculiar to the GENESIS project was the lack of decision on a common platform for development and deployment of the GENESIS environment. This issue also manifested itself in the choice of development tools by each partner. OSCAR in particular was developed exclusively in a Linux environment for much of the project in order to take advantage of the useful Free software tools available such as ready-made if simple version control in the form of CVS and accessory packages such as ViewCVS and StatCVS. The systems used by the OSCAR developers were ignored and perhaps not so readily familiar to the development partners at other institutions. Perhaps as a result of this, the attempt to set up a centralised CVS repository for the whole project failed when most developers ignored the repository and continued using their own methods and tools.

Problems of quality in the external dependencies of OSCAR were also apparent. In particular, the jCVS [8] code used to connect to a CVS pserver installation was poorly documented and difficult to use, resulting in major delays when implementing the CVS interface component and an extensive code review with associated refactoring. Instead, a simple interface class using execution calls to an external CVS binary would have performed the tasks successfully without requiring such a significant coding effort.

Compounding the quality problem, key external dependencies of OSCAR were found not to run under Windows after several months of development. These components included the CVS pserver tool and the PostgreSQL database. Despite this problem, the OSCAR code was flexible enough to be ported to Windows and MySQL very quickly, albeit at the cost of some features and significant expenditure on maintenance of the two separate database back-ends.

Conflicts also arose between the versions of certain dependencies that the GENESIS components had. For example, the OSCAR code relied on a very modern version of the XML parser tools, requiring a complex set-up process to install them where the JDK would use them. The versions required by OSCAR at one point conflicted with the requirements of other components of GENESIS, requiring a work-around.

The most important lesson here is that projects should agree on the components to use and the features they must have, including what platform they should run on if appropriate before development starts in earnest. These decisions should be taken at an early stage, even if the chosen platform is not to the liking of a minority of development partners. In the case of the GENESIS project, Windows would have been a suitable consensus platform as only the OSCAR team preferred Linux as their development environment. If at all possible use of multiple platforms should be avoided as using multiple platforms is unwieldy, especially in a demo session!

Though much interest in the OSCAR concept and our prototype was shown by potential users outside the institutions participating in the GENESIS project, our software dissemination activities were very poorly organised. Presentations at conferences often attracted potential users, andon occasion our web presence elicited interest from companies that were interested in using the system. Therefore, if attracting external users is a goal, we think that the development of an information pack including a gold CD, or at the very least a flyer, should proceed alongside the software development instead of being an activity conducted in the final stages. The information pack need not contain software initially, but a collection of all the papers, presentations and other

useful documentation would be very helpful to potential users. The web presence itself was limited, and the job of maintaining it was never really carried out. In order to attract users, it is key to show that the project is active by having a frequently-updated web presence. As academics, however, we were given credit for our publications, not for our web sites!

Finally, there was the failure to ensure sustained collaboration between partners in the consortium. The OSCAR development team was often excluded from decision making since all the other development partners were located close together in the same country, meaning they could meet frequently to discuss development issues. Much of these discussions and decisions had no impact on the OSCAR component, but this lack of communication did not foster good working relationships. The OSCAR team could only meet the other partners infrequently due to the difficulty of all the partners getting to the same location. Therefore, the bulk of the development partners were able to identify integration risks far faster and earlier than the OSCAR team managed to do. These failings are not attributable to the personnel involved: geographical distance has a deleterious effect on coordination and productivity in software engineering projects [6]. Many open source projects, however, do manage to avoid or work round these problems.

## 3. Resolving Conflict: Our Experiences

Within the OSCAR team the use of unit testing was extremely helpful; they fulfilled the need to test an API and provided comprehensive yet simple example programs for most parts of the system. However, relying on test-cases alone to document one's system is extremely poor practice, whatever the current XP adherents may advocate!

The introduction of elements of XP philosophy (pair programming, API agreement by test cases) and the use of code reviews were found to be effective at promoting communication. The largest subsystem that was reviewed and refactored was the CVS interface code. Other subsystems that were refactored include the Workspace management subsystem, the user session code, the graphical client and the introduction of a logging aspect throughout the OSCAR code.

Components were reselected and repurposed to address issues related to the complex and inefficient implementation code. For example, our custom data-binding code developed to transform artefacts into XML and vice-versa was replaced by the Castor framework to improve quality and flexibility. Also, the meta-data storage component was supplemented with the CVS repository so the unreliability or possible non-availability of the RDBMS would not affect a running OSCAR instance. Thus it became possible to run OSCAR without PostgreSQL if necessary.

Since continuous integration with the rest of GENESIS did not occur, an internal customer for the services of OS CAR for testing purposes was required. The CoDEEDS project needed a data storage component to contain the details of software components, design constraints and decisions taken to satisfy those constraints with the available components in a particular software system [1,2]. Though integration with CoDEEDS identified many problems with OSCAR, these problems caused delays in the development of the CoDEEDS system while its developer waited for bugs in OSCAR to be fixed. Ideally such a collaboration shouldhave begun at the inception of both projects rather than towards the end since otherwise the problem of balancing the ongoing development of each project with the need to maintain stable software or requirements for the other project to work with gave rise to conflicting demands on the OSCAR developers. Balancing the differing needs of each partner is difficult; industrial partners require dependable software to solve problems whilst academic partners wish to explore novel software solutions.

Risk management plans and scheduling are very important for academic projects since their schedules are so tightly constrained that even a single poor design decision can massively impact the outcome of the project. Once again, we advise that sub-projects such as OSCAR play the expectations game; promise very little and attempt to deliver more.

## 4. Conclusions

Our research development work was more or less demand-driven by the other partners implying that software would be exchanged regularly between partners. Unfortunately, release management and thus the prospect of early integration with the other components of GENESIS was poor, partially due to the conflicting configuration management philosophies of the OSCAR team and other development partners. For example, the OSCAR team prepared nightly builds with the intent that the other partners would download the nightly builds to try them each day. However, the other partners preferred a managed release program with software released in full at particular intervals with the bugs in each release fixed independently of the ongoing development effort.

Ideally consensus on the management of the release process or a conscious decision not to bother should have been achieved early in the project to the satisfaction of all parties. Additionally it is unreasonable to expect that small development teams should support more than one stable release of the software at once. The best solution is probably a compromise; a stable release should be maintained where possible and a nightly build produced to track the cutting-edge development. Additionally, the developers using OSCAR should have anticipated the need to be somewhat adventurous rather than expecting all bugs to be discovered, isolated and fixed by the OSCAR developers alone.

The choice of tools in the wider GENESIS project also caused certain conflict and collaboration problems for the developers of OSCAR and other system components. The main tool for development was the web-based SiteScape Forum which, similar to the BSCW environment, allows documents to be uploaded, placed under version control and comments attached to them. Though useful when preparing monolithic documents or similar deliverables, this tool was not suitable for managing source code. Consequently, all the partners developed their own ways of managing the source of their components leading to conflicts and difficulty when releasing software.

Since an effort had been made to provide a single tool for all users, with some success for certain types of artefact, the lack of a suitable configuration management system was an unfortunate omission.

The development of OSCAR was hampered in the main by poor collaboration and communication within the immediate project team and the wider project. The irony of a project dedicated to creating collaborative software being unable to collaborate effectively has not escaped the authors, though the poor support offered by the SiteScape system and e-mail for collaborative code development in contrast to the effective support they offered for collaborative document development is at least partially to blame. Misguided or careless choices of implementation technology also delayed the project extensively. The ready availability of supposedly re-usable Open Source components is a double-edged sword; while they may save implementation time they may also waste it if of poor quality. Finally, the nature of short-term research projects themselves also contributed to the problems; recovering from bad decisions was that much harder as there was little free time or resources to use when repairing the damage.

The largest failing of the GENESIS project was the seeming inability to retain external users, despite a large number of enquiries into the software. In part this may be explained by the lack of maturity of the project's software, but mainly by the fact that dissemination of the software as Open Source was relegated to the end of the project; almost an afterthought. The interest, solicited and unsolicited that GENESIS recieved from users indicates that a real need for this software exists and that the project carelessly failed to take advantage of this.

Interaction with the wider community was consequently limited, even though one of the express goals of the project was to engage the Open Source community with the product. The overall project strategy may have been at fault: initial closed development was intended to "seed" external activity by providing already mature software to the community but instead made the project appear moribund much of the time. Using SourceForge or a similar site from the beginning may have helped visibility and provided better common tool support.

## 5. References


[1] C. Boldyreff and P. Kyaw. A framework for developing a design evolution environment. In Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC). IEEE, 2003. Dallas, Texas.

[2] C. Boldyreff, P. Kyaw, D. Nutter, and S. Rank. Architectural framework for a collaborative design environment. In Proceedings of Second ASERC Workshop on Software Architecture, 2003. Banff, Canada.

[3] C. Boldyreff, J. Lavery, D. Nutter, and S. Rank. Open Source development processes and tools. In J. F. et al, editor, Proceedings of the 3rd Workshop on Open Source Software Engineering, pages 1518, 2003.

[4] C. Boldyreff, D. Nutter, and S. Rank. Active artefact management for distributed software engineering. In Workshop on Cooperative Support for Distributed Software Engineerng Processes, Proceedings of COMPSAC 2002, pages 10811086. IEEE, August 2002.

[5] C. Boldyreff, D. Nutter, and S. Rank. Open source artefact management. In J. F. et al, editor, Proceedings of the 2nd Workshop on Open Source Software Engineering, pages 3 10, 2002.

[6] J. A. Espinosa, R. E. Kraut, S. A. Slaughter, J. F. Lerch, J. D. Herbsleb, and A. Mockus. Shared mental models, familiar-ity, and coordination: A multi-method study of distributed software teams. In Proceedings of the 23rd International Conference in Information Systems (ICIS), pages 425433, Barcelona, Spain, 2002.

[7] M. Gaeta and P. Ritrovato. Generalised environment for process management in cooperative software engineering. In International Computer Software and Applications Conference, volume 26, pages 10491059, Oxford, England, August 2002. IEEE.

[8] The jCVS project. http://www.jcvs.org, 2003.

[9] M. Lanza. Codecrawler: Lessons learned in building a software visualisation tool. In G. Canfora, M. van den Brand, and T. Gym'othy, editors, Proceedings of the Seventh European Conference on Software Maintenance and Reengineering, pages 409418. IEEE Computer Society, 26-28 March 2003.

[10] D. Nutter, S. Rank, and C. Boldyreff. Architectural requirements for an Open Source Component and Artefact Repository System within GENESIS. In Proceedings of the Open Source Software Development Workshop, pages 176 196. University Of Newcastle, February 2002.

[11] E. S. Raymond. The Cathedral and the Bazaar: Musing on Linux and Open Source by an accidental revolutionary. O'Reilly and Associates, 1999.