An Evaluation Framework to Drive Future Evolution of a Research Prototype

David Nutter, Stephen Rank and Cornelia Boldyreff
Faculty Of Applied Computing Sciences
University Of Lincoln,UK
LN6 7TS
{*dnutter,srank,cboldyreff*}*@lincoln.ac.uk*

Abstract

The Open Source Component Artefact Repository (OSCAR) requires evaluation to confirm its suitability as a development environment for distributed software engineers. The evaluation will take note of several factors including usability of OSCAR as a stand-alone system, scalability and maintainability of the system and novel features not provided by existing artefact management systems. Additionally, the evaluation design attempts to address some of the omissions (due to time constraints) from the industrial partner evaluations.

This evaluation is intended to be a prelude to the evaluation of the awareness support being added to OSCAR; thus establishing a baseline to which the effects of awareness support may be compared.

1 Introduction

As part of the GENESIS project [6, 14, 13, 12] the Open Source Component Artefact Repository (OSCAR) [10, 4] was developed and has been used to store software artefacts created by other components of the system, including a work-flow engine, management metrics tool and a project management application. OSCAR is also being used in other projects to provide basic operations on artefacts and consequently requires continuous maintenance and evolution. To assist in directing this evolution, an evaluation of this prototype form of OSCAR is required.

The evaluation discussed here is intended to complement our previous work on evaluating the awareness support being introduced in the OSCAR system [9]. Other than establishing the capabilities of the base OSCAR environment it is not intended to continue that work. Instead, a more focussed evaluation of OSCAR is sought; the evaluation of GENESIS was necessarily broad, as the scope of the GENESIS system demanded a holistic evaluation. In order to direct the evolution of OSCAR, a more thorough understanding of user's reactions to the system than that provided by the industrial evaluation undertaken within the GENESIS project is required as OSCAR and its simpler cousin SAM were "hidden" behind other GENESIS components during that evaluation. It has been noted that evaluation of certain aspects of a software system may itself may be a driver of evolution [2], thus a thorough evaluation of OSCAR may assist in identifying new directions for evolution and re-engineering of the existing code-base, beyond the immediate goals of projects re-using the system.

As a collaborative system, OSCAR has different evaluation requirements from a non-collaborative application. For example, the evaluation must demonstrate that OSCAR improves the relationship and understanding between users over that achieved by existing methods such as email file exchange and telephone. These properties are not easily measured by studying the system alone; user involvement is necessary to obtain the best results. However, certain system properties such as the presence of documentation and usability criteria may be useful as indicators of how good the system will be in a collaborative conttext

2 Industrial Partner Evaluation Results

The evaluation of GENESIS by the industrial partners was performed using a checklist to compare the initial user requirements to the finished platform. The tasks undertaken by each partner were simulations based on their existing practices; one partner was an ERP solution vendor and simulated a typical three-site customisation project, the other used data and developers from an old project to re-run the project and added an observer who would inject events to simulate real world problems. These would include unscheduled activity (deviations from the work plan, feedback in the development process) and other small problems. The partner then evaluated how well the platform dealt with these occurrences.

The industrial partners' evaluation studied several aspects of the finished GENESIS platform. Firstly, both part-

ners confirmed that their **Process management requirements** were addressed by the work-flow system, however risk management facilities were missing and one partner had difficulties changing an instance of a running process. Secondly, one of the partners confirmed that the multiple methods (local and remote) of accessing the toolset worked correctly, thus addressing their **Accessibility** requirements for remote working etc. Thirdly, both partners confirmed that the **Resource Management** application worked according to their expectations.

Certain aspects of the system (ease of installation, flexibility, learnability, reliability and a few others) were assessed in a purely subjective manner by both partners. This was mainly due to time constraints and the difficulty of quantifying these nebulous traits. It was therefore necessary to rely on user impressions rather than rigorous analysis.

Though the GENESIS platform supported metrics calculation based on the workflow model and items in the artefact repository, neither of the partners had identified any **Metrics requirements** in the simulated projects. Though such requirements did exist in other projects, and thus the metrics functionality is likely to be used if GENESIS was deployed fully in the partner organisations, this omission meant the metrics component of the GENESIS platform was not evaluated.

Unfortunately, the OSCAR component was deemed "not evaluable from the users' point of view" since it was not directly accessible by the platform's users. Consequently the following evaluation framework has been developed to address this omission.

3 Evaluation Design

Before any modifications beyond corrective maintenance are made to OSCAR an evaluation of the current utility and quality must be conducted, in order to establish a baseline against which any perfective modifications may be compared. This is to ensure that improvement in utility commensurate with the effort expended has been provided.

Firstly, the **quality** of the original code is important for future modifications. If much time must be spent performing corrective maintenance, less time will be available for perfective and adaptive maintenance. Related to quality, **maintainability** aspects such as the pertinence of the documentation and the state of the software support suite including test-cases is important for promoting developer comprehension. If OSCAR is difficult to maintain, it will be likewise difficult to add new features to it. Maintainability is also linked to usability: if a system is not used, no feedback to drive the maintenance effort will reach the developers.

Secondly OSCAR intends to provide **novel features** currently not offered by traditional artefact or configuration management systems. Any evaluation must show that these features exist and furthermore, are useful to developers or as a basis for additional funtionality.

Thirdly the **scalability** of the system is important for large groups of users though for small groups (of, say, 1–5 people) this may not be a problem. Scalability and performance are aspects of **usability**, the most important aspect of which from a user point of view is the usability of the stand-alone client, as this will be the first point of contact with the OSCAR system an ordinary user has. When OSCAR was hidden behind the other components of GENESIS, whether this client was suitable or not was not as important. As the only general-purpose interface to the stand-alone OSCAR system, the usability must be thoroughly examined before modifications are made to the interface.

To be successful, future modifications must lead to improvement in one or more of the above areas and, perhaps more importantly, lead to no significant deterioration of the existing capabilities. The evaluation criteria discussed below will be applied to the modified system versions to verify this. To this end, the Goal Question Metric (GQM) [1] approach has been selected to perform this evaluation. However, while the original GQM [16] methodology focusses on measurement of the software process and its products, our model is intended to lead to direct improvements of the product and process. The information gained from applying "true" GQM models is often used for just this task, our methodology merely removes an extra step.

Since a large user community is not available to question or examine, this baseline evaluation will necessarily be limited to using more passive techniques or tolerating small, biased, sample sizes. Therefore, at this stage of the evaluation semi-automated evaluation of code quality and maintainability [15] will be the major focus of evaluation, with a simple checklist [7] and metrics-based [11] examination of the client's usability according to accepted best-practice in user interface design. The possibilities for future evaluation of the baseline code are discussed below in **Operational Evaluation**.

3.1 Quality Aspects

The system possesses several quantifiable quality aspects. Though no scale has been defined—since metrics may be applicable to some aspects more than others—the closer the score to zero (no problems) the better:

Comprehension A poorly structured program is difficult to maintain, especially by other people than the original authors. Lots of issues with complexity and poor modularity may lead to an unmaintainable legacy

Existing problems These would include known defects, the presence of outdated documentation and a consistently high defect rate.

Process Problems with the development process would be indicated by outdated or missing documentation and test-cases, poor version control, a consistently high defect rate for that class or subsystem and ownership issues. Several defects in the process that produced OSCAR have already been identified [5] and care must be taken to ensure these do not recur.

Usability Visual clutter, inconsistent terminology *etc.* make it hard for users to learn to use the system and may discourage them from persevering with it.

3.2 Maintainability Evaluation

The maintainability of the system will be a decision based on the following metrics and assessment criteria. Entries marked with an asterisk (throughout the rest of this paper) rely on qualitative interpretation of the source data rather than other techniques:

Defect density The defect density of code is a good indicator of code quality. The lower the density of defects (per thousand lines of code), the better the code quality. Defect density may vary within software systems depending on quality and complexity of each of the individual subsystems, though work[8] has been undertaken to address this difficulty in quantification.

Complexity Looping and other decision making constructs can make comprehension difficult if they are too large or too deeply nested. It is generally better to increase call depth rather than adding more inner loops, for example.

Call depth Deep call structures reduce the ease of program comprehension and debugging. The call depth should be reduced where possible, but not at the expense of modularity.

Modularity Tightly-coupled multi-function classes should be avoided where possible as they make comprehension and debugging more difficult.

Module size & duplication Large modules should be avoided where practical, but not at the expense of adding tight coupling to other classes. Use of utility packages to abstract common functionality from the rest of the system should be employed.

Ownership issues* If certain parts of the system are only understood by one developer, this should be addressed.

Test cases and documentation The presence of test cases and other support tools are indicative of a verifiable level of functionality and thus quality. A check to see if the documentation is pertinent or obsolete should be made. If obsolete, comprehension is penalised.

3.3 User Interface Evaluation

In addition to the criteria above, the user interface will be evaluated using several special criteria. During the development of OSCAR, two developers undertook a qualitative review of the user interface as a precursor to pair-based further development of the interface. During this process, several common problems in the interface design were discovered and this is reflected in the following specialist criteria. As before, an asterisk indicates qualitative interpretation:

Consistency Consistency of terminology between different parts of the client must be checked. Synonyms for operations lead to user confusion and should be eliminated. Additionally, misleading names for functionality (subjective decision) should be avoided. A glossary of terms should be readily available.

Clarity Labels and explanatory text should be well lined up. Misalignment or obscured information will lose points. Additionally, use of special purpose dialogs should be kept to a minimum (*e.g.*, for file selection). Where one should use a dialog is a subjective decision, however.

Succinctness (clicks-per-operation) If the client requires too many individual commands for common operations (check in, check out, edit meta-data etc) it will be cumbersome to use. Similarly, if common individual commands are difficult to access (in menus etc) the client will be similarly cumbersome. Presence of keyboard shortcuts a bonus.

Extensibility* Related to maintainability, the interface must be extendable. Recent experience in the GeniSOM [3] indicate that this system is extensible; allowing the addition of a new workspace view based on a map visualisation.

Nesting depth The "depth" of screens, menus and dialogs is important. If they are nested too deeply the user may get lost. However, a tradeoff must be made with visual clutter.

Application integration The ease of integrating external applications such as editors and viewers should be examined. Since the client can't do everything itself, integrating external software should be easy.

Visual clutter* For the GUI client only. The more widgets there are per screen, the more difficult the client is to comprehend at first glance. Thus, unused functionality should be hidden away. However, this requirement must be balanced with that of nesting depth.

Goal	*Purpose*	Improve
	Issue	the usability of
	Object	the client application
	Viewpoint	User
Question		Is the interface consistent and clear?
Metrics		Instances of inconsistent terminology Instances of poor layout
Question		Is the user interface responsive and accessible?
Metrics		Regular operations accessible in < 2 clicks All operations accessible in < 6 clicks Subjective impression of responsiveness Nesting depth < 3
Question		Does it integrate well with other applications?
Metrics		Time taken to integrate five arbitrary applications
Goal	*Purpose*	Improve
	Issue	the quality of
	Object	the OSCAR back end
	Viewpoint	Developer
Question		Is the test coverage sufficient?
Metrics		Total modules - modules without test cases
Question		Is the system too complex?
Metrics		(automated) nesting depth (automated) coupling and module size (automated) call depth
Question		Is the code consistent?
Metrics		(automated) style checker
Question		Is the system extensible?
Metrics		Count of facia classes/interfaces Count of plugins
Goal	*Purpose*	Assess
	Issue	the maintainability of
	Object	the OSCAR code base
	Viewpoint	Developers (current/new)
Question		Is it well structured?
Metrics		(automated) call depth Manual check for duplicates (automated) modularity
Question		Is the test suite sufficient?
Metrics		Simple test coverage, as above.
Question		Is it well documented?
Metrics		Instances of outdated/incomplete docs Number of undocumented classes

Table 1. Sample of the GQM model for OSCAR

Goal	*Purpose*	Improve
	Issue	the success of
	Object	the development process
	Viewpoint	Developer/Manager
Question		Is the documentation pertinent?
Metrics		Instances of outdated/incomplete docs
Question		Are the requirements fulfilled?
Metrics		Checklist survey of requirements
Question		Are all parts of the system understood by 2 or more developers?
Metrics		Manual survey
Question		Is the defect rate declining?
Metrics		Count fix commits in CVS log over time
Goal	*Purpose*	Improve
	Issue	the deployability
	Object	of OSCAR
	Viewpoint	User
Question		How much time is required for system management?
Metrics		Time to perform routine maintenance (cleanup) Time to restore system from crash
Question		Is the install process well documented?
Metrics		Subjective impression of document
Question		Is the system quick to install?
Metrics		Timed install

3.4 GQM Model

Table 1 shows the initial GQM model for this software system. The quantitative aspects of the system are associated with particular goals and used to generate various questions to be answered using metrics or a qualitative assessment of the system.

The metrics output and qualitative interpretation will be used to develop a maintenance plan for OSCAR which will attempt to remove any extant maintainability problems and focus future work on areas that need attention. Below, the future evaluation design is discussed, detailing the steps needed to evaluate OSCAR's utility to users in greater depth, rather than OSCAR's utility to future developers.

Furthermore, additional problems spotted during the evaluation (which necessitates some study of code and ancillary software artefacts alongside automated metrics generation) but not part of the overall model will be corrected by the developers. If these problems are both detectable by measurement and likely to recur, the model will be expanded to include them. Thus the model will evolve along with the OSCAR system as it develops.

Due to space constraints and the need for thorough discussion of the methodology, a typical set of results from an evaluation run is not presented here. As we refine the evaluation model and improve the software, we will publish a "typical" result set with discussion of the issues and implications of the results.

3.5 Operational Evaluation

Any future evaluation will be based on the findings of the GENESIS project industrial partner's evaluation, brief findings of which were discussed earlier. This evaluation was high-level and ignored the possibility of running OSCAR as a stand-alone system. In order to build on the existing results, a similar approach has been selected to complement the GQM; that of a user questionnaire derived from some of the questions that are difficult to answer by using metrics.

The purpose of the expanded evaluation will be twofold: gauge the user response to the OSCAR concept of meta-data enriched, "active" artefacts and to verify that OSCAR is useful to developers performing everyday tasks, something that the preliminary evaluation cannot determine. However, evaluation of any modifications (such as the addition of awareness support) beyond the basic OSCAR system will be undertaken in the context of those projects, not the evolution of OSCAR.

The target user base will be a small number of research students initially, and potentially a number of partners in wider research collaborations that we are involved in. In addition, OSCAR will be re-used in a number of other research activities aside from this ongoing evaluation.

4 Conclusions

Extensive evaluation of OSCAR was not possible in the short timeframe of the GENESIS project, though ensuring usability and thus maintainability is absolutely necessary for the ongoing successful evolution and use of OSCAR in further projects. The problems identified in the process that developed OSCAR originally [5] may recur without ongoing software measurement and evaluation; indeed the lack of regular evaluation was mooted as a probable cause of problems such as user misunderstandings and design defects. Given plans to integrate the prototype version of OSCAR with other projects, evaluation has been deemed necessary.

The framework discussed here is intended to provide a lightweight, ongoing evaluation of the basic OSCAR code in order to drive the maintenance effort and ensure that the software's quality and usability do not degrade over time. This framework does not perform many of the tasks found in more thorough evaluations such as direct user involvement and observation. However, this lightweight evaluation can be conducted very quickly by a single developer with the help of readily available software tools. In contrast, a thorough evaluation would take considerably longer, involve more people and would require a user base willing to be studied on a regular basis.

The necessity and desirability of conducting more rigorous evaluation of the system on occasion was alluded to earlier in this paper, for example when user-focussed evaluation of OSCAR extensions such as awareness is carried out. To this end the future evaluations involving users discussed earlier will be tailored to the circumstances of the occasion. We believe that an evaluation of this sort, where possible, will provide sufficient high quality evaluation data to indicate that OSCAR is capable of directly supporting collaboration. Due to the lack of user involvement in the GQM-based evaluation discussed in this paper, the results obtained from it will necessarily be of limited use in directly assessing the success of the collaborative properties of the system. However, general software quality can only help support the development of these features.

References

[1] V. R. Basili, G. Caldiera, and H. D. Rombach. *Encyclopaedia Of Software Engineering*, chapter The Goal Question Metric Approach. Wiley, 1994.

[2] C. Boldyreff. Determination and evaluation of web accessibility. In *Proceedings of WETICE 2002*, pages 35–41. IEEE Computer Society, June 2002.

[3] C. Boldyreff and J. Brittle. Self-organizing maps applied in visualising large software collections. In A. V. Deursen, C. Knight, J. I. Maletic, and M.-A. Storey, editors, *Proceedings of the 2nd IEEE Workshop on Visualising Software for Understanding and Analysis*, pages 99–104. IEEE, September 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 1081–1086. IEEE, August 2002.

[5] C. Boldyreff, D. Nutter, and S. Rank. Communication and conflict issues in collaborative software research projects. To appear in 3rd Open Source Software workshop, ICSE 2004, March 2004.

[6] M. Gaeta and P. Ritrovato. Generalised environment for process management in cooperative software engineering. In *International Computer Software and Applications Conference*, volume 26, pages 1049–1059, Oxford, England, August 2002. IEEE.

[7] J. Harvey, editor. *The Evaluation Cookbook*. Learning Technology Dissemination Initative, 1998.

[8] Y. K. Malaiya and J. Denton. Module size distribution and defect density. In *Proceedings of ISSRE'00*, pages 62–71, San Jose, California, October 2000. IEEE Computer Society.

[9] D. Nutter and C. Boldyreff. Historical awareness support and its evaluation in collaborative software engineering. In *Proceedings of WETICE 2003*, pages 171–176. IEEE Computer Society, June 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] R. S. Pressman. *Software Engineering: A Practitioners Approach*, chapter Software Process and Project Metrics, pages 79–112. McGraw-Hill, 5th edition, 2001.

[12] P. Ritrovato. Generalised enviroment for process management in cooperative software engineering. In *Workshop on Cooperative Supports for Distributed Software Engineering Processes, Proceedings of COMPSAC2002*, pages 1049–1053. IEEE, August 2002.

[13] P. Ritrovato. IST project fact sheet: GEneralized eNvironment for procEsS management in cooperatIve Software engineering. http://istresults.cordis.lu/, March 2004.

[14] P. Ritrovato. Open solution to managing distributed software developments. http://istresults.cordis.lu/, March 2004.

[15] N. Truong, P. Roe, and P. Bancroft. Static analysis of students' Java programs. In R. Lister and A. Young, editors, *Proceedings of the Sixth Australian Computing Education Conference (ACE2004)*, volume 30 of *Conferences In Research and Practice in Information Technology*, 2004. Dunedin, New Zealand.

[16] R. van Solingen and E. Berghout. *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*. McGraw Hill, 1999.