To appear in Proc. Workshop on Empirical Research in Software Testing (WERST 2004).

A Software Engineering Research Repository

Roger T. Alexander James M. Bieman Robert B. France
Software Assurance Laboratory
Computer Science Department
Colorado State University
Fort Collins, Colorado USA
[rta,bieman,france]@cs.colostate.edu

ABSTRACT

Software developers lack objective information to assess the effectiveness of current and proposed technologies and practices. We are developing a Software Engineering Research Repository (SERR), a widely-accessible repository of software development artifacts. The core artifacts of SERR are development artifacts, for example, code, models, and test cases, organized by projects. These raw materials provide a base for carrying out analyses and the results can be stored as related artifacts in the repository. In this way, the knowledge content of the repository can be built incrementally, providing a rich support base for other research activities. Research programs can thus build upon the results produced by other programs that utilize the repository. Software developers will be able to access objective data from the repository to assess software engineering tools and techniques.

1. INTRODUCTION

While much progress has been made in moving software development towards an engineering discipline, advances in development methods, tool environments, languages, and management techniques have not kept pace with the growing complexity of software solutions. Software development organizations are increasingly being challenged to develop high quality software solutions in a timely and cost-effective manner.

Developers faced with the task of developing large, complex software solutions (e.g., secure, flexible, fault-tolerant distributed systems) often use mostly anecdotal information to help select software development methods and tools. A lack of scientifically-based data makes it difficult to (1) assess the extent that current technologies and practices address software development problems, and (2) identify and understand fundamental software development problems.

Significant progress can be made if a widely-accessible repository of software development artifacts with well-known properties and characteristics is created. Such a repository can be used to support community-based evolution of software development knowledge. The artifacts can provide the data needed by research concerned with understanding development phenomenon (e.g., understanding the effects of change on system architectures). The results of these studies can then be added to the repository, thus enhancing the role of the repository as a software development knowledge source. The repository can, for example, (1) facilitate comparative analyses of experiences related to the development of systems, (2) be used to communicate software development successes and failures to the software development community, (3) provide raw data on system artifacts (e.g., designs, code), technologies, and practices to research programs, (4) support efforts related to collecting empirical data about software processes, development technologies and notations, implemented systems and their models, and (5) support technology transfer and education. Currently there are no widely-accessible repositories that we know of that have the variety of software design information that we envisage.

The fact is that we do not understand the characteristics of the population of professionally developed computer programs. Thus, our ability to draw conclusions from scientific results and generalize them is severely hampered. Historically, this problem is a result of the closed nature of professional software development. With few exceptions, companies with significant investment in intellectual property are reluctant to make their source code available to the scientific community. Thus, software engineering researchers are forced to use small samples that are usually drawn from student populations. However, the proliferation and adoption of open source software offers the possibility that this may change.

2. OVERVIEW OF THE SERR

Our goal is to develop a sustainable, widely-accessible repository of software development artifacts to support software development research and education. The repository is referred to as the *Software Engineering Research Repository* (SERR). The core artifacts of SERR are development artifacts, for example, code, models, and test cases, organized by projects. These raw materials provide a base for carrying out analyses and the results can be stored as related artifacts in the repository. In this way, the knowledge content of the repository can be built incrementally, thus providing a rich support base for other research activities. Research programs can thus build upon the results produced by other programs that utilize the repository.

While users of SERR may get the impression that artifacts are physically stored in a central location, this may not be the case for all artifacts. We will allow SERR contributors to locate artifacts on their sites and build links to the sites from the SERR site. The intent is to build up a network of software development research resources that researchers and practitioners can access from a single point. We also envisage that the repository will contain other artifacts that may not necessarily be directly related to development projects in the repository, for example, development experience reports, tool

evaluations, and links to software engineering research programs.

The SERR system hardware will consist of front-end and back-end servers, an array of hard disks, and back-up equipment and media. SERR software will support information retrieval using a data base, and customized software to support the submission of artifacts to the repository, and specialized analyses of artifacts. Such analyses may be conducted using SERR software and SERR hardware. Some analyses will be conducted using client hardware and/or software.

3. BENEFITS OF THE SERR

Through proper management and quality participation the SERR can evolve into a software development knowledge repository that is an indispensable resource for local and national research programs. Access to such a repository can speed the rate of advances in software development by reducing the need for research programs to develop base information and knowledge needed to support their research goals.

3.1 Software Engineering Education

The repository is a sustainable research environment that will also provide significant benefits to software engineering education. The contents of the SERR can improve the knowledge and skills that software engineering students are able to bring to a competitive marketplace. The SERR will contain designs of large-scale systems with related analysis and other information that can be accessed for classroom use. Thus, students can experience, first-hand, some of the complexities associated with the development of large systems. Specifically, the SERR can provide students with industrial-strength software artifacts that can lead to a deeper understanding of the issues and problems surrounding software development in industry. This will make their educational experience much more realistic and will better prepare them for the workplace.

3.2 Research.

Clearly the SERR will benefit empirical software engineering research in a variety of areas including model-based software development, business modeling, code analysis, formal verification and testing, software evaluation, software reuse, and domain-specific software engineering. This repository will enhance research capabilities by providing needed information and data.

3.3 Benefits to the Development Community

Practitioners have trouble obtaining good exemplar software artifacts to help guide their work. The SERR can serve as a source of experiences, exemplar artifacts, and benchmark artifacts for evaluation of technologies, for the overall software development community. We also see a distinct possibility that some of the tools developed could become commercial products, which could generate revenue that could help sustain future evolution of the SERR.

3.4 Long Term Impact and SERR Evolution

The repository will be a long-term entity, with value for many years. Although computer hardware becomes obsolete quickly, software persists. When software is upgraded, the older version (or portions of older versions) remains in the new version. A key research concern is how systems evolve. We are not concerned about having the latest software versions.

The repository will be a knowledge base that will grow over time. Many of the items in the repository will be *conceptual* entities (e.g., analysis and design models) rather than code. These conceptual entities can have a very long "shelf life", especially if they are technology-independent. The contents will not be static; the repository will continually be re-seeded and will reflect an accumulation of knowledge about developing systems in different domains.

As more organizations contribute and/or participate, the greater the value of the repository to these and other organizations. For example, the repository can become a test bed for software reuse studies, and for the development of tools to support reuse. Many organizations would like to know how to make reuse work better and to try new tools to retrieve reusable artifacts, however they do not have the time or human resources to do this internally.

4. THE REPOSITORY INFRASTRUCTURE

A conceptual model of the repository infrastructure is shown in figure 1. The round edge boxes represent data storage and management resources (e.g., databases), the rectangular boxes represent subsystems, the dashed directed lines indicate access relationships among subsystems and the storage mechanisms. The stick characters represent actors (i.e., roles that are played by systems or humans outside of the SERR), and the solid lines represent associations between the actors and the SERR subsystems. SERR

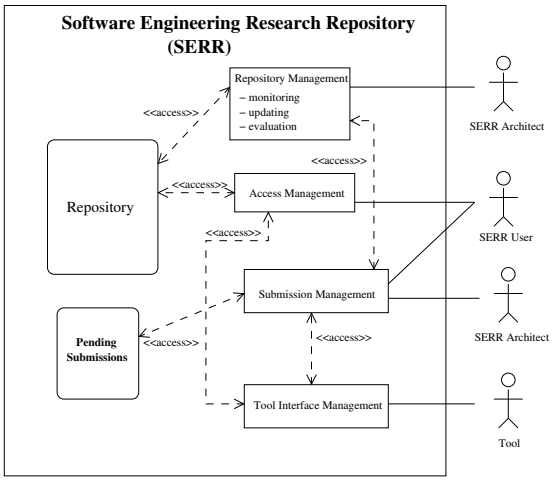

Figure 1: Conceptual SERR Architecture

Users (clients) will access repository artifacts through the *Access Management* subsystem. This subsystem will provide a web-based interface to the repository, and will have a back-end that controls all client access to the repository. SERR users will also be able to submit candidate artifacts to the SERR through the *Submission Management* subsystem. Submissions are stored in the *Pending Submissions* repository until they are evaluated by the Repository Architect. If the submissions are of sufficient quality they are entered into the repository through the *Repository Management* subsystem. The Repository Management subsystem will allow the architect to (1) package, classify and insert new or modified artifacts, (2) delete artifacts and (3) update links between artifacts. This subsystem will also automatically collect usage data and generate regular reports, thus allowing the architect to monitor and track usage of the repository. The *Tool Interface Management* subsystem provides the interface that client-based tools will use to retrieve artifacts from, analyze, and submit artifacts to the SERR.

The SERR will also have collection tools that use search engines and appropriate filters to identify potential items for the repository. An appropriate database will store the items.

The repository will initially consist of the following types of artifacts: (1) system models and implementations, (2) data and other analysis artifacts obtained by analyzing models and implementations (e.g., data dependency, control flow, and call graphs derived from code, and measures of coupling and cohesion derived from design models), (3) process information, and (4) miscellaneous reference information.

The core artifacts of the repository will be system development artifacts (e.g., models, test cases, and implementations) organized around projects. A project is a collection of development artifacts for a system. The scope of a project can vary from a stand-alone application system that addresses a single problem domain within an organization (e.g., a payroll system), or a software utility such as a word-processor or operating system, to a large integrated system of systems that addresses enterprise-wide application needs (e.g., an order processing system that integrates customer handling, order processing, and accounting systems).

4.1 Model and implementation artifacts.

A project in the SERR is a structure of artifacts related to the development of a system. This includes all code, model, and other artifacts (e.g., test cases, project plans), and all versions of all artifacts. Projects are intended to provide views of systems at different levels of abstraction. Traceability, refinement, and realization relationships are defined among the artifacts. Techniques developed to support project repositories for an industrial collaborator (see [4]) will be utilized.

SERR projects provide practitioners and students with realistic examples of the types of models that can be used to represent software requirements and designs, and their relationships with code implementations. Researchers can use these artifacts in studies that require analysis of software development artifacts. Requirements and design models in the SERR will include those expressed in standard design notations, such as the Unified Modeling Language (UML) [3]. It is expected that the novelty and growing popularity of the UML will drive demand for sample models that illustrate how the UML can be used to model systems at various levels of abstraction. The development of a standard XML representation (called XMI) and tools for translating between tool-based representations of UML diagrams and XMI representations allows us to store UML diagrams in an accessible form.

4.2 Analysis artifacts.

The analysis tools will identify the design structures, and then quantify the components in terms of the roles that the component plays in the design structure. We will develop our research tools primarily by extending existing research tools and commercial object modeling tools.

4.3 Process Information.

Process information includes business processes as well as the effort required to produce versions of software, defects found, and change histories. This information is commonly collected by software development organizations. However, organizations are often reluctant to release such information publicly. We have been successful in generating valuable process data (e.g., software change histories) from common source code control system logs.

4.4 Miscellaneous reference information.

Information that is not directly tied to projects stored in the repository will also be accessible through SERR. Examples of such information are industry experience reports, and links to sites containing information on software research programs and research reference information. In this respect, the intent is to evolve the SERR so that it can act as a starting point for accessing information and publicly available software development and research resources.

5. USING SERR TO SUPPORT RESEARCH

The repository can support a number of software engineering research activities. We outline a few of the research activities that can benefit from the existence of the SERR.

5.1 Developing design evaluation technology.

The choice of software design structures determines, in part, whether a software system will be easier or harder to test and maintain. Current technology offers only rough guidelines to help designers make good choices; there are few objective mechanisms for evaluating design quality. Data in the form of software design information is needed to demonstrate relationships between design choices and external quality attributes such as software adaptability and reusability. The repository will provide the data. System design documents, descriptions of planned evolution, and software repair records are SERR artifacts that can be especially useful to this type of research.

5.2 Investigate system evolution.

Useful software systems evolve as a result of changes in technology or changes in system environments. Evolving large, complex software systems using current development technologies is difficult, error-prone, and expensive, as was evident in Y2K efforts. Currently, industries are struggling to evolve their large, integrated application systems to gain competitive advantage, in the face of rapidly changing technologies (e.g., Internet/web-based technologies). Our ongoing research on software evolution is described in [1]. The repository will contain evolution trails of software artifacts that will be used to study how software systems evolve. The results of this study will be used to develop engineering approaches to software evolution that manage the complexity and cost of evolving large, complex software systems.

5.3 Document the empirical value of models.

Information system developers use models to help in the development process. Conceptual models, logical models, implementation models, and test models are created at various points in the process. Most developers agree that models provide value. However, assertions of the value of models are based on conjecture and anecdote, not on evidence gathered from empirical studies. The SERR can a vehicle to collect and document empirical data regarding the value of models — the costs and benefits of creating and maintaining various types of models. This data will allow researchers, students, and practitioners to better determine how and when to use models.

5.4 Investigate model-based software development.

Models of software requirements and designs can help manage the complexity of creating and evolving large, integrated software systems. The repository will provide artifacts that will allow us to study the precise relationships between models and technology-specific technologies (e.g., MicroSoft's .Net, Sun's J2EE, C++, C). Results of this study will be used to develop model-based approaches to software development in which models are used to generate implementations in particular technologies [2]).

5.5 Document enterprise models.

Enterprise information models include models that show the information that a business processes (entities) and the interrelationships between these entities, and the processes followed in performing their business functions. Most examples that practitioners, researchers, and teachers have available are very limited in scope, with most encompassing only very small aspects of real business problems. One could collect, document, and store more complete enterprise-wide models, including both high-level conceptual models as well as low-level design models, as well as ones in between in addition to actual production code and test suites. Researchers will then be able to analyze various models to support, for example, empirical work or to identify reusable artifacts such as patterns (e.g., see [2]); students will gain a clearer understanding of models that are used in real-world information system development, and gain a larger business-wide view of IT; and practitioners will have examples to use to compare with their own models and architectures.

5.6 Develop Cost Models.

Businesses aim to select IT solutions that provide the most benefit for their cost. There are almost always many options, and it is usually not easy to understand the tradeoffs that come with each option. A research project might develop and store business cost models for various types of solutions to various types of business IT problems. Researchers, students, and practitioners can then compare business models and their proposed solutions, and better determine which solutions are better for their needs.

6. REFERENCES

[1] R. B. France and J. Bieman. Multi-view software evolution: A uml-based framework for evolving object-oriented software. In *Proceedings of the International Conference on Software Maintenance 2001*, 2001.

[2] R. B. France, S. Ghosh, and D. Turk. Towards a model-driven approach to reuse. In *Proceedings of the 7th International Conference on Object-Oriented Information Systems (OOIS 2001)*. Springer, 2001.

[3] The Object Management Group (OMG). Unified Modeling Language. Version 1.3, OMG, http://www.omg.org, June 1999.

[4] R. Trask and R. France. RIGR - a repository model based approach to management. In *Proceedings of UML Workshop on the Practical UML-Based Rigorous Development Methods*. GI-Edition, Lecture Notes in Informatics, 2001.