

A Formal Model of System and Software Engineering Experience

Douglas S. Lange

SPAWARSYSCEN

D44207

53118 GATCHELL RD RM 426C

San Diego, CA 92152-7420

dlange@computer.org

Valdis Berzins

Naval Postgraduate School

Computer Science Dept. Code CS/Bv

833 Dyer Road

Monterey, CA 93943-5118

berzins@cs.nps.navy.mil

Abstract

This paper describes some of the issues involved in formalizing a previously fuzzy aspect of software development – the use of experience. This paper reports the status of work in progress and raises some issues for thought.

Introduction and Motivation

Software and systems engineering organizations need to improve their performance from project to project. Such improvement can only come about from understanding and analyzing the experience gained during past projects. Since organizations often outlast the membership of individuals within that organization, and since collaboration among many individuals is often required, automated storage, retrieval, and decision support should be useful. Efforts to develop engineering experience bases in the literature focus on databases of artifacts, preserving free text correspondence, and developing structured metadata for lessons learned documents. Our position is that these are not sufficient models for an engineering experience base because they do not capture the semantics of the experiences at a level that can be effectively used by decision support algorithms.

The field of artificial intelligence has provided approaches that can be used to associate the conditions that led to previous decisions, the decisions themselves, and the measured and analyzed results. Similarly through applications of modal logic, elements of experience that seemingly contradict each other can coexist in an experience base and provide useful insight to decision makers. We propose that a more formal logic-based model is essential in order to build an experience base that can provide substantive computer aid to support engineering process improvement.

Background

This section surveys and assesses relevant previous work.

Organizational Memory

Organizations can be seen as functionally resembling information-processing systems that process information from the environment. As such, organizations exhibit memory that is similar in function to that of individuals. The components of the system include: [WalUng91]

- Sensors, which act to receive information
- A processing capability that processes information using defined symbols
- A memory where information is sent for storage and from which it can be retrieved.

Individuals in problem-solving and decision-making activities acquire information. Organizational memory comes about due to the sharing of information among individuals in different ways. Organizational memory is stored information about a decision stimulus and response from an organization's history that the organization can use to help make present decisions. [WalUng91] This definition points out a critical factor in defining organizational memory that has been missed by experience base researchers. Without information about the decision stimulus, the "why" of an organizational response cannot be determined. Walsh and Ungson argue that only if the decision stimulus is retained can the experience be used to meet the requirements of more novel situations, and that without it one is likely to promote deleterious decision making.

Continuous Process Improvement

IEEE 1220 is among the standards available for system engineering concerns to use in defining their engineering process. According to the standard, within the policies and procedures of the project, the continuous improvement of products and processes must be addressed. Among the activities recommended by the standard for accomplishing continuous improvement of processes are:

- Maintaining a self-assessment program to determine the maturity of the enterprises systems engineering practices, and
- Capturing the lessons learned on each project and incorporate them into enterprise training courses, as appropriate, to improve the application of the SEP.

IEEE 1220 goes on to state that in order to be compliant and perform systems engineering at a standard industry level, one must have a means to capture the experiences generated by undertaking projects. [IEEE 1220] The issue we wish to explore is what kinds of formal models would be appropriate for that purpose.

Experience Bases

Software and systems engineering experience base research can be divided into three categories. First are previous efforts to improve the use of largely unstructured lessons learned artifacts. A second category uses the tools available in free-text applications such as email and chat capabilities. Third are database applications. These are either based on relational or object databases.

Lessons Learned

Lessons learned documents are among the least structured objects in which experience can be preserved. It is not surprising therefore that research in improving the management of lessons learned documents would seek to impose some structure on them. Birk and Tautz describe a process for packaging lessons learned that has as its basis a quality control mechanism and a structuring of the document [BirTau98]. The structure used divides a lesson-learned document into four sections.

- Object The software artifact that the lesson is concerned with. It can be a tangible product, or something less tangible such as a process or method.
- Context Describes the situation within which the problem and solution are relevant.
- Problem The problem that is being solved. This along with the solution is the core of the lesson learned.
- Solution The solution to the problem. This along with the problem is the core of the lesson learned.

Birk and Tautz go on to classify types of lessons learned through a semantic network of presupposed relationships among the types of lessons learned and their relationship to situations and artifacts. Other studies support the importance of classifying lessons in order to impose a structure that allows lessons to be retrieved. Statz creates elements for lessons, each of which can have multiple attributes [Sta99]. These form the equivalent of facets for searching a lessons learned database.

Free-Text Applications

There is no faster, less intrusive way to capture experience than to record the conversations of the people involved in an effort. At its simplest, this approach can take on the form of a running logbook. An example of this approach uses a distributed running diary as the experience base. The diary serves to allow process participants to record problems and solutions in free text form. Search facilities allow users to go back and look at previous entries [Rob+00].

Another approach to capturing free text is found in the Answer Garden [Ack98]. Answer Garden attempts to augment organizational knowledge in two ways. First, Answer Garden makes answers to common questions available through structured and unstructured user interactions. Second, it provides information about who in an organization is considered an expert in a particular area. Answer Garden provides a hierarchical structure to help users find the answers to frequently asked questions. The user interface asks a series of questions. The answers drive the user along a tree structure towards an answer. If the problem to be solved is new and an answer cannot be found, the application prompts the user to describe the problem and it is sent to an expert. The expert answers the question and can then place it and the answer into the hierarchy so that future users can find it. Free text search engines also allow the user to find answers directly.

Database Applications

Databases provide a natural platform on which to store the experiences of an organization. When designing an experience base using such a tool, the foci of the effort are on what types of information will be collected and how will a user find the information needed to solve a particular problem [Alt+99] [BroRun99] [Wan+99]. The primary deficiency of database applications for experience bases is that they store only artifacts and data. These can either be the results of the project or the results of the packaging of experience in the form of new process definitions. The rationale and the reasoning behind the experience is missing. This makes it difficult for the recipient to generalize and determine whether or not the experience is applicable. The contributions of these research efforts are mostly in the area of information retrieval.

Assessment

The common weakness of all three approaches to experience bases is that they passively store data and rely on the users to perform all interpretation, analysis, and adaptation. Better formal models that can support some aspects of the intended use of this data are needed.

Evolution Control Systems

Evolution control systems provide a hint at a path to provide better formal modeling and decision support. All of the standards mentioned above require that projects keep track of information that allows them to adequately control their processes. Such information includes:

- Requirements,
- Schedules,
- Defects,
- Process descriptions, and
- Process metrics.

The configuration management process [SEI99] requires that a project be able to establish and track baselines, manage changes to baselines, and be able to perform audits tracing changes to and from engineering decisions.

The current state of the practice in this area involves using configuration management tools to help manage this information. Similar practices exist for other process areas. The state of the research involves the use of formalized process definitions that allow more powerful automated evolution control systems to be developed. These systems can form the basis around which tools supporting

As defined in the Relational Hypergraph Model of Software Evolution (RHMSE) software evolution consists of two main sub-processes. First is the software prototype evolution process and second is a software production generation process [Har+99]. The figure below illustrates the evolution lifecycle. Throughout the prototyping process, the elements communicated are requirements, software prototypes, demonstrations, and user criticisms [Ber+97].

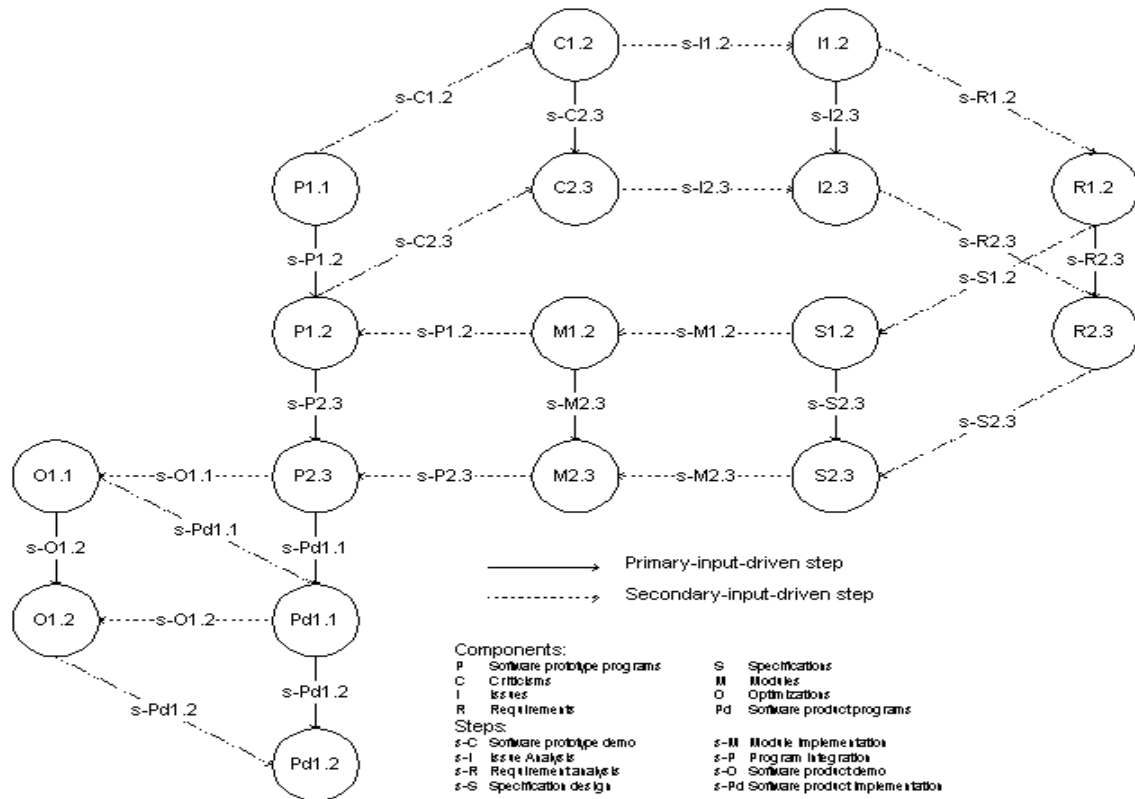


Figure 1. Software Evolution Hypergraph

Several of the elements necessary for experience bases can come directly from an evolution control system. The information tracked as well as the semantic network representation of the evolution history provide context points to which problems and solutions can be attached.

Why a Formal Model?

Process improvement is by nature an act of inference. In order to determine how to improve a process, an analyst must use knowledge of the previously employed processes, information about the environment in which the process has been executed, and the measurements taken during execution. Problems and their solutions figure highly not only in revising documented processes, but also in solving unanticipated problems as they arise. For organizational memory to be used in novel situations, decision makers must be able to *generalize* from previous experience to encompass the current problem, as they perceive it.

Perception too argues against simple collections of artifacts. Not only must the experience base be able to present experience in light of different environments, but the beliefs of the engineers about the environment and the results of their efforts must be modeled. Two engineers may come to different conclusions based on their differing beliefs and these differences can in themselves provide valuable insight.

The ultimate argument for a formal model is to make automation of information collection and knowledge inference possible. Information vital to an experience base is

difficult to collect. It is time consuming and does not directly relate to, what most engineers feel, is the job at hand. By representing experience in a formal model, and in particular one that can be represented by a semantic network, a linkage to an evolutionary control system such as the one described above becomes possible.

Expected Characteristics

The following characteristics are expected to drive the form of the model being developed.

- Experience is a collection of beliefs about what has happened in the past and what the situations were that were relevant to the outcomes. Modal operators will be necessary. Modality should allow assumptions of monotonic behavior (within each modality) to be accepted. Without modality, non-monotonic examples are likely.
- More than one point of view may exist about what the situations or results were concerning a previous project. One, both, or neither may be correct. The points of view must be linked to the corresponding supporting evidence to support judgments of which point of view is most relevant to a given problem.
- Generalization cannot be done without the beliefs concerning the environment that existed during previous projects.
- Time may or may not be relevant to the results experienced. One may not know whether it is relevant until attempting to generalize or compare to a current situation. Temporal operators will be necessary.
- Vital components of the experience base will be directly extracted from evolution control constructs. Examples of this are the requirements set, task assignments, personnel qualifications, actual and predicted labor and task duration, test results, demonstration plans, and user feedback in the form of criticisms.
- The use of a semantic network for the model rather than linear logic descriptions will allow easier integration with a human computer interface.
- The use of modal logic will allow easier integration with a standard agent architecture [FIPA].

Issues

The following issues are expected to arise during the model's development.

- Experience can change one's beliefs about past situations. This will likely cause a problem.
- We will need to show that information is entered into the system without onerous effort on the part of the engineers. It will be difficult to decide how much effort is appropriate. It is believed this would vary from organization to organization and that it will therefore be difficult to study. However, statistical data that shows values within two standard deviations might be a useful result.

Example

To examine the rationale behind our belief that a logic-based model of experience is needed, the expected characteristics of the model, and the issues anticipated, the following scenario is analyzed.

Scenario

A test team is putting together a test plan and procedures to support version three of a large software project. Two years of modest fixes and updates occurred between versions one and two, and two years have also occurred since version two was released. The last test plan was written nearly three years ago.

When version one was tested, a large effort was made to exhaustively examine the application programmer interface (API) set of the major server components in the system. Many third party applications as well as large parts of the software being tested make use of these services and it was felt that this was an important risk reduction strategy. When version two was tested, no API testing was conducted.

No members of the version three team were part of the API test effort. Neither were any of them part of the decision processes that resulted in testing the API in version one and not testing them in version two.

A Database Solution with Case Based Reasoning for Information Retrieval

Two test plans are in the experience base. Based on the solution offered by [Wan+99], there is information of the following nature associated with each test plan.

Attribute	Test Plan for Version 1	Test Plan for Version 2
Organization	C2 Systems Engineering	C2 Systems Engineering
Staff size	15	10
Application domain	Command and Control	Command and Control
Improvement goal	Reliability	Reliability
Programming Language	C	C
Software System Size	2000 KLOC	2500 KLOC

Table 1. Attribute Data for Artifacts

Which should be retrieved? Using [Wan+99], this would depend on our goals. Our goal is to create a new test plan. Our current staff size is eight, and the current system size is 3000 KLOC, so version 2 will come out as the closest match. Was this the right decision? There may be an artifact in the system that is a lesson's learned document describing the decision process if somebody wrote one. This would take us to the approach in [BirTau98] with retrieval being case based.

Our Approach

Our approach involves using the information created through the use of an evolution control system. The decisions made are represented in such a system through the assignments created, the criticisms levied against a system, and the issues and requirements managed [Ber+97]. The relationships between objects in such a system need to be expanded however. [Ber+97] limited the relationships to *poses*, *affects*, *supports*, *depends_on*. The relationships between decisions and the supporting information must be recorded

When users began to notice problems with the services, the following criticisms would be posed. They are shown in the form of triples alongside text that describes some of the content of the information nodes.

User1, poses, Criticism1
Criticism1, affects, Version1
User1, poses, Criticism2
Criticism2, affects, Version1
User2, poses, Criticism3
Criticism3, affects, Version1
...

As criticisms are entered, analysts are assigned the task of constructing and associating issues and requirements to the criticisms. The analysts' issues and the relationships they find among them are recorded in the evolution control system. An experience base can use them beyond requirements tracing. Each analyst's entries illustrate something about their beliefs concerning the state of the project. Therefore, when *Analyst1*, decides that the services and their associated API are not well enough tested, she creates an issue and associates all of the criticisms that she believes are symptoms of that problem to the issue. Further analysis indicates that the issue is related to the requirements that created the services in question and their related API.

Criticism1, affects, Issue1
Criticism2, affects, Issue1
...
Criticism1000, affects, Issue1
Issue1, affects, Requirement2
Issue1, affects, Requirement3
...

Further *support* for *Analyst1's* belief can be found in schedule changes recorded in the evolution control system. Schedule slips corresponding to the difficulties that developers

are having with the services form part of the rationale for the project manager to create a requirement for full API testing.

StepState223, updates27, StepState156

updates27, reason3, rationale15

updates27, supports3, Issue1

...

The experience base adds decision nodes (among others) to the features tracked by the evolution control system. The decision to create a requirement for API testing follows from the support created by the large set of criticisms all relating to a single issue and by the schedule slips that have been associated to the same issue.

In version two of the system, the defects did not appear, because no new service creation requirements were assigned. Basically reuse of the services with a few repairs was the decision. Therefore no requirement to test the API set in version two was created.

It is now time to plan the testing for version three. New requirements similar to those in version one are created. The experience base agents infer that these new requirements that also include interface components could also support the belief that API testing is an important feature for the test plan. If no new interface requirements were added to the system the inference would not be supported and the agent would not recommend the API testing.

In order for this inference to occur, a basic model of features and relationships of software engineering projects must be present in an ontology service available to the agents. Information such as interface requirements being a special form of requirement will need to be "known" by the agents. The semantics of the relationships will need to be defined in more detail than currently used by evolution control systems, but their current relationships will map into the ontology without affecting their capabilities.

Conclusions

Through a more formal model of software engineering experience, human and automated inference can be improved in support of engineering decision-making. An equally important benefit will be the ability to link the experience base to an evolution control system. In this way, the collection of experience becomes a side effect of the normal management decision process.

References

- [Ack98] Ackerman, M., "Augmenting organizational memory: a field study of answer garden", *ACM Transactions on Information Systems*, Vol. 16. Issue 3, 1998.

- [Alt+99] Althoff, K., Birk, A., Hartkopf, S., Müller, W., Nick, M., Surmann, D., and Tautz, C., "Managing Software Engineering Experience for Comprehensive Reuse", Proceedings of the Eleventh International Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Germany, 1999.
- [Ber+97] Berzins, V., Ibrahim, O., Luqi: "A Requirements Evolution Model for Computer Aided Prototyping", Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering, Madrid, Spain, 1997.
- [BirTau98] Birk, A. and Tautz, C., "Knowledge Management of Software Engineering Lessons Learned", Proceedings of the Tenth International Conference on Software Engineering and Knowledge Engineering, San Francisco Bay, California, USA, 1998.
- [BroRun99] Broomé, M. and Runeson, P., "Technical Requirements for the Implementation of an Experience Base", Proceedings of the Eleventh International Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Germany, 1999.
- [FIPA] Foundation for Intelligent Physical Agents. <http://www.cselt.stet.it/fipa/>.
- [Har+99] Harn, M., Berzins, V., and Luqi, "Computer-Aided Software Evolution Based on a Formal Model", Proceedings of the 13th International Conference on Systems Engineering, Las Vegas, NV, USA, 1999.
- [IEEE1220] IEEE Std 1220-1998, *IEEE Standard for Application and Management of the Systems Engineering Process*, Institute of Electrical and Electronics Engineers, 1998.
- [Rob+00] Robinson, M., Kovalainen, M., and Auramäki, E., "Diary as Dialogue in Papermill Process Control", *Communications of the ACM*, Vol. 43, No. 1, January 2000.
- [SEI99] Software Engineering Institute, Capability Maturity Model®-Integrated-Systems/Software Engineering: Staged Representation - Volume 1, Version 0.2b, 1999.
- [Sta99] Statz, J., "Leverage Your Lessons", *IEEE Software*, Vol. 16. No. 2, IEEE, 1999.
- [WalUng91] Walsh, J. and Ungson, G., "Organizational Memory", *Academy of Management Review*, Vol. 16., No. 1, January 1991.
- [Wan+99] Wangenheim, C., Althoff, K., and Barcia, R., "Intelligent Retrieval of Software Engineering Experienceware", Proceedings of the Eleventh International Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Germany, 1999.