Funk P.J. and Crnkovic I., Reuse, Validation and Verification of System Development Processes. Proceedings to the First International Workshop on the Requirements Engineering Process, Florence, Italy, IEEE Computer Society, September 1999.

Reuse, Validation and Verification of System Development Processes

Peter J. Funk and Ivica Crnkovic
Mälardalen University, Computer Science Lab
Västerås, Sweden, {peter.funk, ivica.crnkovic}@mdh.se

Abstract

Large companies often use standardized template development processes. Project-specific adaptation of templates must address aspects such as: project resources (time/ staff), standards, regulations, etc. Adapting templates is a particularly manual process requiring skill and for large companies represents a large proportion of the total development cost. Integrating locally gained experience and updating the template process is tedious work and resources for such updates are rarely available. Fortunately, formal representation of processes and process components enables reuse, analysis and comparison of processes and parts of processes. We use a case-based reasoning (CBR) approach which permits identification and reuse of processes or parts of processes. The formal notation allows the user to sketch new processes or adapt template processes. These sketches/ adaptations are used in a matching process which identifies and suggest the reuse of similar processes and tasks stored in the library. Once an adaptation has been successfully used, it is automatically added to the case library.

1. Introduction

System development is one of the most complex processes found in organizations today [1, 2]. Large companies often use a standard development process as a framework for all their development projects. Such a best practice process aims at reflecting the company's collective experience, its commitment to quality and minimum lead time and reflects the category of projects and the level of skill of those engaged in its projects. An explicit system development process is rarely used and in small projects and success or failure is mostly dependent on the individual skill and experience of the project leader and the project members. Far to often such skill and experience is acquired through unsuccessful projects. The majority of software projects in industry fail [3] and one of the main reasons for failure is believed to be the inadequacy of in software development processes. Many technology-intensive companies have tens of thousands of people working in different projects, and the savings achieved by using a uniform system development process throughout the company is believed to be considerable when compared with permitting each project manager to develop his own process. It enables the company to ensure that new projects meet internal and external requirements such as quality standards, control over progress and checkpoints to identify problems as early as possible etc.
Unfortunately, reality in large companies often requires extensive local adaptation of template processes to suit different types of projects and different circumstances in the environment of the project. If project-specific modifications and adaptations are made, the resulting process cannot be warranted to meet the overall demands of the company and customers and there is a risk that less suitable adaptations may be introduced causing subsequent difficulties in the project. Additionally, it is difficult to transfer the experience gained from local adaptations back to the template processes and difficult to spread it within the company. Collecting the experience from adaptations performed locally requires skilled staff (often short in supply) and much time and effort as the overall experience is distributed over many people, each with a small part of the total experience. The experience gained from completed projects is seldom collected. There are strong indications that the assembly of experience gained from completed projects to be reused in improving the outcome of future projects may be one of the means of building corporate memories [4, 5, 6].

2. Development Processes

The wide variety of abstract system development methodologies available includes the waterfall and V models. These models are often too generic and need careful adaptation to suit specific types of project, company stan-

dards and the skill level of the employees. Companies therefore often develop their own detailed system development template to meet internal and external requirements. These template processes are then adapted to specific projects. The information concerned is mostly available in informal manuals with guidelines, quality and control points and examples. They are difficult to use and there is commonly no or very little support provided to assist those engaged in projects in following these documents and guidelines.
System development processes used are usually represented informally. Recent technology and tools which support the production and analysis of system development processes are beneficial. Standards such as ISO9001 and models such as CMM (Capability Maturity Model) can be incorporated in the tools to aid the production of system development processes which meet these requirements. These tools unfortunately rarely aid the transfer between users of knowledge gained from local adaptations. A case-based reasoning approach is proposed in which processes are adapted and stored in a case library for reuse in part or in whole. A matching algorithm identifies similar, but not necessarily exactly identical processes in the case library.

3. Case-Based Process Tailoring

The CABS system (Case-Based Specification System, see section 4 on case-based reasoning) formalizes processes based on graphical examples and uses a temporal logic for internal formal representation. The user outlines project circumstances, and sketches process examples. The matching algorithm uses the input to identify similar processes and parts of processes and proposes these for reuse. The CABS system was originally developed for behavioral requirements specifications [7, 8] such as the behavior of telephone features. CABS treats a process definition task as an experimental development task (see Figure 1) and arrives quickly at something we can validate and verify in a variety of ways. These sketches are then refined, compared with similar process descriptions in the case library and used to identify parts for reuse or to point out differences. This will aid the users of CABS to refining and extending the system development process until they are convinced that the requirements for the specific project are met. CABS is currently being modified to fit more closely the application domain of system development processes. The internal representation and matching algorithm is the same as used in [7]. The formal representation is based on predicate logic (transition rules) and has sufficient expressive power to represent system development processes, but is not unnecessarily expressive, "a formal representation should be as simple as possible, but no simpler." [9]. Limiting expressiveness is a major approach to taming the combinatorial explosion in production systems [10]. We do not confront the user with the formal notation and the notation is concealed behind a user interface.

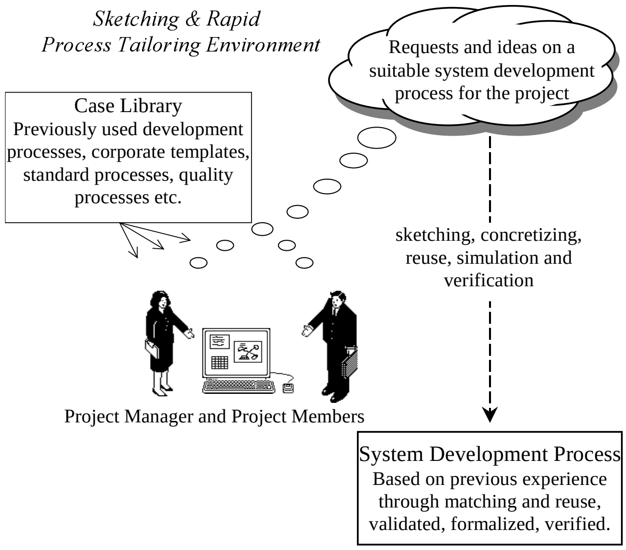

Figure 1: From an idea to a formalized process definition.

4. Case-Based Reasoning

The central concept of case-based reasoning is expressed by Riesbeck and Schank as: "the essence of how human reasoning works. People reason from experience. They use their own experience if they have a relevant one, or they make use of the experience of others" [11]. Aamodt and Plaza´s picture, Figure 2, illustrates the main ideas of case-based reasoning: a problem is presented in the top left corner, similar cases are retrieved from a case library and the most suitable case is selected and re-used. It may be necessary to revise the most suitable case for it to solve the problem. If the solution is approved, the problem and its solution are stored in the case library. The next time a similar problem is encountered, less adaptation of the retrieved case may be needed and the process will be simplified if similar problems are often encountered and the features identifying similar cases are sufficiently recognizable.
The study of a previous case in which a similar problem has been solved may, in some situations, aid the process of finding a solution because a case provides a context for understanding [12]. A case-based system may also adapt to changing demands, for example, if a new type of problem not previously encountered is solved (if no similar cases are available, a solution of the problem is most likely to be produced manually). The problem solved and

its solution are stored in the case library as a new case, with the aim of expanding its competence [13]. The next time the system encounters the same or a similar problem, the system will have increased its potential to produce a solution. It is more likely that, in a rule-based system, it would be necessary to update the rules to include this new class of problem.

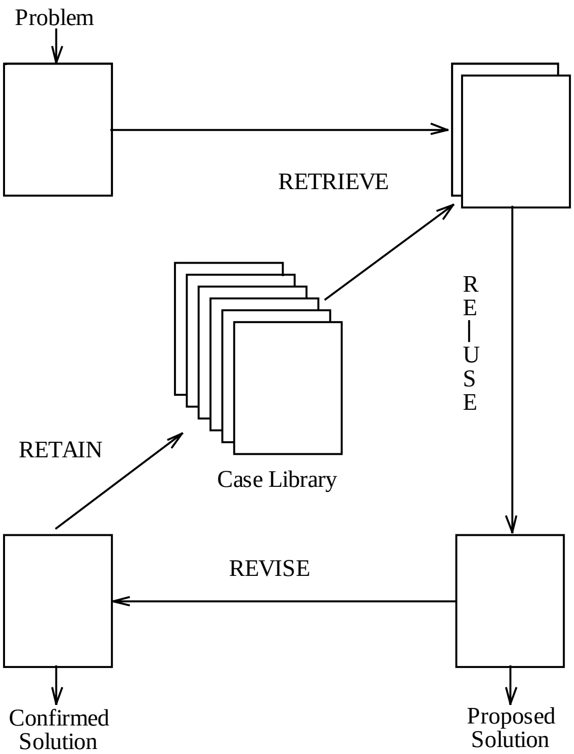

Figure 2: General architecture of a case-based reasoning system. Adapted from Aamodt, Plaza [13].

Case-based reasoning may be suitable for problem areas in which the knowledge of how a solution is created is poorly understood [14], e.g. the creation and adaptation of system development processes. In technical domains, case-based reasoning has been applied to a variety of a application domains such as: architectural design support [15]; qualitative reasoning in engineering design [16], [17], software specification reuse [18], software re-use [19], fault correction in help desk applications [14], building regulations [20], business modeling [21], fault diagnosis and repair of software [22]. There are already certain CBR systems in commercial use and CBR components embedded in other systems.

In summary, case-based reasoning may be applied to application domains which are not sufficiently well understood to create a consistent and complete knowledge-base for solving the problems automatically, provided that:

- problems and their solutions have similarities.
- a case library with past problems and their solutions is available or can be created.
- solutions can be adapted and re-used for similar problems.
- there are suitable means of identifying relevant cases in the case library.

We suggest that development processes fit these requirements well if a formal notation is used. The user does not need to know that there is a formal notation involved and draws the process descriptions in a graphical editor as usual, the editor translates the diagrams to the formal textual representation used in matching and analysis. A matching algorithm which identifies similar behavior is used [7].

5. Representation of Development Processes

A system development process is modeled by a set of tasks (e.g. design subsystem, specify function, test function, formally verify function, verify system, handle release, …). Each task is either atomic and cannot be divided into smaller parts or it may be defined by a number of more specific tasks. The chosen granularity of a development process depends on factors such as the magnitude of the project and the experience level of project members. Process components may be sequentialized or concurrent if parallel or incremental development is applied in the project (verification will identify dependencies causing problems if performed in parallel, e.g. if one process component needs output from the other process component). Tasks and their input, output and work description are well defined and may be under CM management [23]. Examples of input and output (see Figure 3) are: validated function requirements; function test plan; implementation proposal, tested code; formally verified code; etc. The ontology for the application domain must be determined carefully [24] as all cases in the case library will be based on these and both reuse and identification of similar processes and tasks is based on this terminology. How to determine an ontology is beyond the scope of this paper. System checkpoints may be defined as a collection of information in a given revision state, for example "milestone 14 is defined as a set of output being in the completed or implemented state".

An example of a task with three needed inputs and two output results is given in Figure 3. The only requirements for the performance of the task are the input and the satisfaction of the necessary selection criteria (the availability of selected tools and skill in using them, manpower needed for task, etc.). Additional information such as a work description (how to produce the output given the input) may be informal text, links to other documents or a process description or workflow description. Input and output are given as terms with arguments (predicate logical formulae used in planning, matching, verification and simulation) and are either defined with other terms or are atomic and defined by informal text.

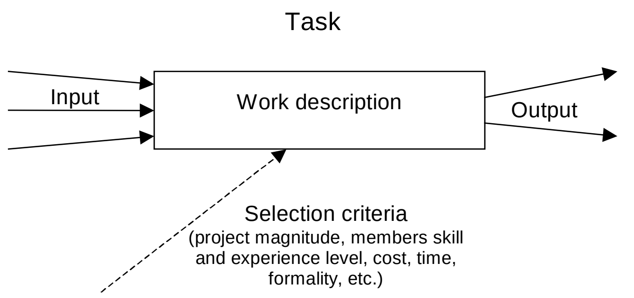

Figure 3: Example of a task

A report or document in this context can be defined as a set of tasks and output in a specific revision state (started/ ongoing/ implemented/ internally approved/ customer approved/ completed/…) in addition to layout information.

6. The CABS Approach to Reuse and Verification

The system is illustrated in Figure 4. In the upper left-hand corner, the user outlines the project, available resources and perhaps sketches of parts of the development process. A description of the project include magnitude, quality requirements, standard requirements, preferred tools etc. The matching algorithm [7] (the second box from the upper left corner) uses input sketches to identify tasks and processes which show similarities with the input requests. After the matching algorithm has identified a set of tasks, this result is used to rank the system development processes stored in the case library. The user is presented with the ranking result and can study the different proposals. When the user selects a proposal, the selected proposal can be validated and verified against the input (the *Revise* box in Figure 4). For example there may be requests and project circumstances not well handled in the proposal and the user may decide to perform certain modifications of the proposed process.

During verification the proposal or modified proposal may not meet quality standards and its shortcomings will be pointed out. If the user rejects the final proposal more examples are requested (broken line out from *Revise* box). If the solution is accepted (after verification and simulation of its behavior), the new or adapted system development process is stored in the case library. Once different tasks in the process have been completed, their outcome, problems and advantages may be added to the task in the library. This gives every task in the case library a track record which may be valuable for use as reference material for future projects in which these tasks are considered for reuse.

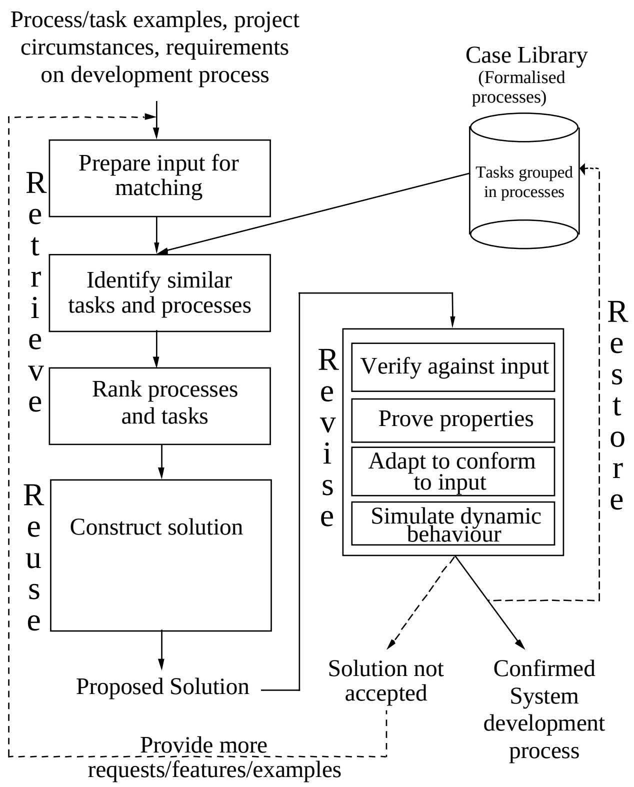

Figure 4: Outline of the CABS approach

7. Conclusion

Using case-based reasoning in combination with formalized system development processes offers certain advantages over current practice in which system development processes are mostly informal. The formalization permits matching and comparison between different development processes. Differences between a particular development process and other processes such as a template process (a high quality process template, a standard medium size software project template, etc.) can be identified. The prime advantage is that successfully adapted system development processes become available for reuse. Adapted system development processes which have been successfully used in a project are automatically made available for reuse (less successful tasks and processes may also be kept in the case library to assist in avoiding similar less successful processes in future). This enables an organization to preserve locally gained experience in terms of improved and adapted system development processes.

The case-based reasoning system (CABS) is currently being adapted to fit the application domain of development processes more closely. In our further work we also need to formalize a number of realistically sized system development processes and store them in the case library. Thereafter we propose an evaluation based on specific projects for which the user need to develop a system development process.

References

[1] Doheny J.G. and Filby I.M. (1996). A framework and Tool for Modeling and Assessing Software Development Processes, The European Software Control and Metrics Conference, 1996.

[2] Doheny J.G. and Filby I.M. (1996) Modelling Software Development Processes and Standards, Technical Report AIAI-TR-205, University of Edinburgh.

[3] Sommerville I. (1996). Software Engineering, fifth edition part one & five, Addison Wesley.

[4] Althoff K.-D., Birk A., Gresse von Wangenheim C.and Tautz C. (1998) CBR for Experimental Software Engineering. In Case-Based Reasoning Technology: From Foundations to Applications. Lenz M., Bartsch-Spörl B., Burkhard H.-D., Wess S. (eds). Springer.

[5] Althoff K.-D., Bomarius F., Tautz C. (1998) Using Case-Based Reasoning Technology to Build Learning Software Organizations. In workshop on Building, Maintaining and Using Organizational Memories. http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/vol-14/ Abecker A., et. Al (eds).

[6] Henninger S. (1997). Applying Organizational Learning Techniques to Software Process Engineering Environments, in proceedings Automated Software Engineering ASE´97.

[7] Funk, P.J. and Robertson D. (1995). Case-Based Selection of Requirements Specifications for Telecommunications Systems. Second European Workshop on Case-Based Reasoning, Proceedings, Keane M., Haton J. P., Manago, M. (eds.), Chantilly, France, pp 293-301.

[8] Funk, P.J. and Robertson D. (1998). Graphical Input Sketches for Producing Formalised Behavioural Requirements. Proceedings of International Workshop on Visualization Issues for Formal Methods, VISUAL´98. April, Lisbon, Protugal.

[9] Zave P. and Jackson M. (1996). Four Dark Corners of Requirements Engineering. ACM.

[10] Acharya A. (1994). Scaling up production systems: Issues approaches and targets. The Knowledge Engineering Review, vol 9:1.

[11] Riesbeck C. and Schank R. (1989). Inside Case-Based Reasoning, Lawrence Erlbaum Inc. Intelligence, Budapest, Hungary, John Wiley & Sons Ltd, pp 390-394.

[12] Kolodner J. (1993), *Case-Based Reasoning*. Morgan Kaufmann.

[13] Aamodt A. and Plaza E (1994), Case-Based Reasoning: Foundational Issues, Methodological Variations and System Approaches. AI Communications, vol 7, pp 39-59.

[14] Watson I. (1997). Applying Case-Based Reasoning: Techniques for Enterprise Systems, Morgan Kaufmann.

[15] Pearce M., Goel A.K., Kolodner J.L., Simring C., Sentosa L. and Billington R. (1992). Case-Based Design Support. IEEE, October, pp 14-20.

[16] Sycara K.P., Navinchandra D., Guttal R., Koning J. and Narasimhan S. (1992). CADET: A Case-Based Synthesis Tool for Engineering Design. International Journal of Expert Systems, vol 4, no. 2, pp 167-188.

[17] Nakatani Y., Tsukiyama M. and Fukuda T. (1992). Engineering Design Support Framework by Case-Based Reasoning. ISA Transaction, vol 31, no. 2, pp 235-180.

[18] Maiden N.A.M. and Sutcliffe A.G. (1995). Requirements Engineering by Example: an Empirical Study. Proceedings of IEEE International Symposium on Requirements Engineering, pp 104-111.

[19] Fouqué G. and Matwin S. (1993). Compositional Software Reuse with Case-Based Reasoning. Conference on Artificial Intelligence Applications 1993, IEEE, Florida.

[20] Yang S.-A., Robertson D. and Lee J. (1995). Use of Case-Based Reasoning in the Domain of Building Regulations. Topics in Case-Based Reasoning, Springer-Verlag, pp 292-306.

[21] Chen-Burger J. and Robertson D. (1998). Formal Support for an Informal Business Modelling Method. 10th International Conference on Software Engineering and Knowledge Engineering (SEKE'98), USA.

[22] Hunt J. (1997). Case based diagnosis and repairs of software faults. Expert Systems, vol 14, no 1, pp 15-23.

[23] Crnkovic I., Funk P.J. and Larsson M. (1999). Processing Requirements by Software Configuration Management. Euromicro 99 Conference, York, June.

[24] Uschold M., (1996). Building Ontologies: Towards a Unified Methodology, Proceedings of Expert Systems 1996, Cambridge, UK.