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 a large proportion of the development cost. Integrating locally gained experiences 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 process components 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 [1a, 1b]. 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 companies collective experience, their commitment to quality and minimum lead time and reflects the category of projects and the level of skill of those engaged in projects. In small projects an explicit system development process is rarely used and the success or failure is mostly dependent on the individual skill and experience of the project leader and the project members. Such skill and experience is expensive to win if they are the result of unsuccessful projects. More than half of all software projects fail [2] and failure is believed to arise from deficiencies 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 here own process. It enables the company to ensure that new projects meet internal requirements in addition to external requirements such as quality standards, control over progress and checkpoints to identify problems as early as possible.

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 overall process cannot be warranted to meet overall demands of the company and customers and there is a risk that less suitable adaptations are introduced which may cause subsequent problems in the project. Finally, the experience gained from local adaptations is difficult to transfer back to the template processes and difficult to spread within the company. To collect the experience from adaptations performed locally require 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.

2. Development Processes

There is a wide variety of abstract system development methodologies available such as the waterfall and V models. These models are often too generic and need careful adaptation to fit specific types of project, company standards 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 thereafter 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 producing and analyzing system development processes are beneficial. Standards such as ISO9001 and models such as CMM (Capability Maturity Model) can be incorporated in the tools and 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. We propose a case-based reasoning approach in which processes adapted and successfully used, are stored in a case library and enable reuse of these processes in part or whole. A matching algorithm identifies similar, but not necessarily exactly identical processes in the case library. These may be reused in part or in whole. A case-based approach can be considered to be an important factor in building and maintaining a corporate memory.

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 sketches process examples and the system proposes previously specified processes or parts of previously specified processes which may be reused. The CABS system was originally developed for behavioral requirements [3] 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. All this in a tightly integrated environment with few restrictions on order or sequence. This will aid the users of CABS to refine and extend the system development process until they are convinced that the requirements for the specific project are met. We have begun making CABS more generic in order to include system development processes as a suitable application domain. 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.” [4]. Limiting expressiveness is a major approach to taming the combinatorial explosion in production systems [5]. We do not confront the user with the formal notation and the notation is concealed behind a user interface.

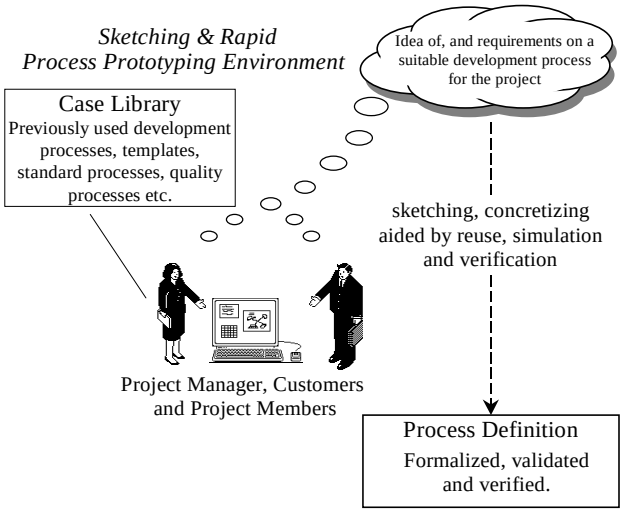

Figure 1: From an idea to a formalized process definition.

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" [6]. Aamodt and Plaza´s picture, Figure 2, illustrates the main ideas of case-based reasoning: a problem is given in the top left corner, similar cases are retrieved from a case library and the most suitable case is selected and re-used. The most suitable case may need to be revised 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 which has solved a similar problem may, in some situations, aid the process of finding a solution because a case provides a context for understanding [8]. 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 [5]. 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 problems.

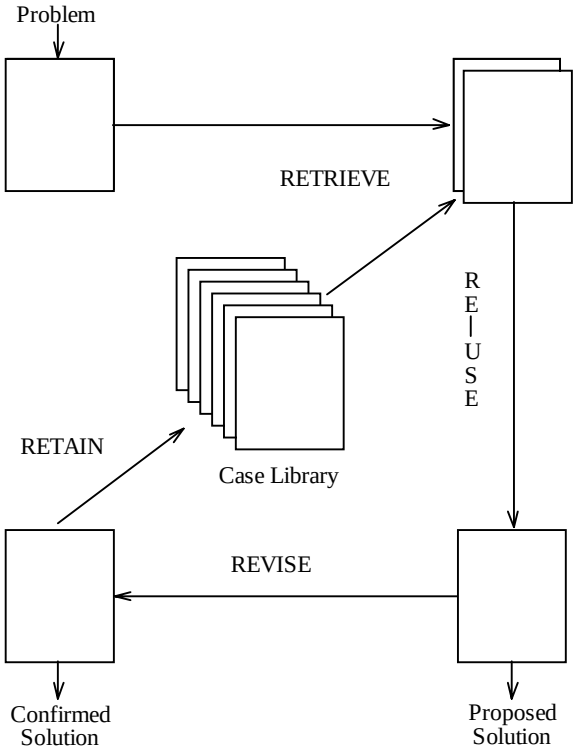

Figure 2: General architecture of a case-based reasoning system. Adapted from [7,Aamodt, Plaza 94].

Case-based reasoning may be suitable for problem areas in which the knowledge of how a solution is created is poorly understood [9], e.g. the creation and adaptation of system development process. In technical domains, case-based reasoning has been applied to a variety of a application domains such as: architectural design support [10]; qualitative reasoning in engineering design [11], [12], software specification re-use [13], software re-use [14], fault correction in help desk applications [9], building regulations [15], business modeling [16], fault diagnosis and repair of software [17]. 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 in how to solve the problem 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 for 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 [3].

5. Representation of Development Processes

A system development process is modeled by dividing the process into a number of process components (e.g. System study, design subsystem, specify function, test function, formally verify function, verify system, handle release, …). Each process component is a set of tasks having a clearly defined input and output. One or more process components can be combined in a system development process. Process components may be sequentialized or concurrent if parallel or incremental development is applied in the project (verification will identify if there are dependencies causing problems if performed in parallel, e.g. if one process component needs output from the other process component). Input and output from process components are well defined items which may be under CM management [18]. 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 to be determined carefully [19] as all cases in the case library will be based on these and both reuse and identification of similar processes and process components is based on input and output items. Atomic items cannot be divided and are the smallest parts reflecting the granularity of the system in which the development process is described. 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 "mile stone 14 is defined as being all output items in the completed or implemented state".

A process component may have one or more tasks each representing a specific activity requiring specific input and producing specific output. An example of a process component with two tasks is given in Figure 3. Individual tasks may be applied in parallel or sequenced in any particular order with the only restriction that tasks only can be performed if the input and the necessary resources are available (resources which are not properly modeled in this example, may be available or selected tools, estimation of manpower needed for task, etc.). In the example in Figure 3 the task is activated if the process component "formally verify function(X)" is requested. All tasks having their requirements met (input available) will be performed. Process components may have more information than shown below but *name*, *input* and *result* is the

information used to model the process. Additional information such as 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.

Process component: formally verify function(X)

Task 1a

Input:

formalized function
requirement(X, internally approved)
formal notation(X, predicate logic)

Result after completion:

formally verified function requirement(X, completed)

Work description/workflow/process/tool:

Use theorem prover for predicate logic to formally verify behavior against original requirements.

Process component: formally verify function(X)

Task 1b

Input:

formalized function
requirement(X, informally approved)
formal notation(X, Petri-Nets)

Result after completion:

formalized function requirement(X, completed)

Work description/workflow/process/tool:

Use Petri Net tools and perform model checking etc. If liveness is proven (absence of deadlock) the task is completed.

Figure 3: Example of two process components

Reports or documents may be defined as a specific set of items and their revision state (started/ ongoing/ implemented/ internally approved/ customer approved/ completed/…) together with layout information showing how to display or print the report.

6. The CABS Approach to Reuse and Verification

The system is illustrated in Figure 4. In the top left-hand corner, the user gives examples of desired parts of the system development process. Examples can be given as graphical examples in which parts of the desired process are exemplified as partial process sketches. Input may be in the form of examples created by the user selecting process components believed to be suitable for the project. Size, quality requirements, standard requirements, preferred tools etc. may also be given as input which may be used to further improve the matching process (not handled in CABS). The matching algorithm [3] (the second box from the top of the left corner) uses the input sketches to identify tasks and process components which show similarities with the input examples. After the matching algorithm has identified a set of tasks and process components this result is used to rank the system development processes stored in the case library. The user is presented with the ranking result and may explore the different proposals. When the user selects a proposal the selected solution can be validated and verified against the input (the *Revise* box in Figure 4). If the user rejects the final solution more examples are requested (broken line out from *Revise* box).

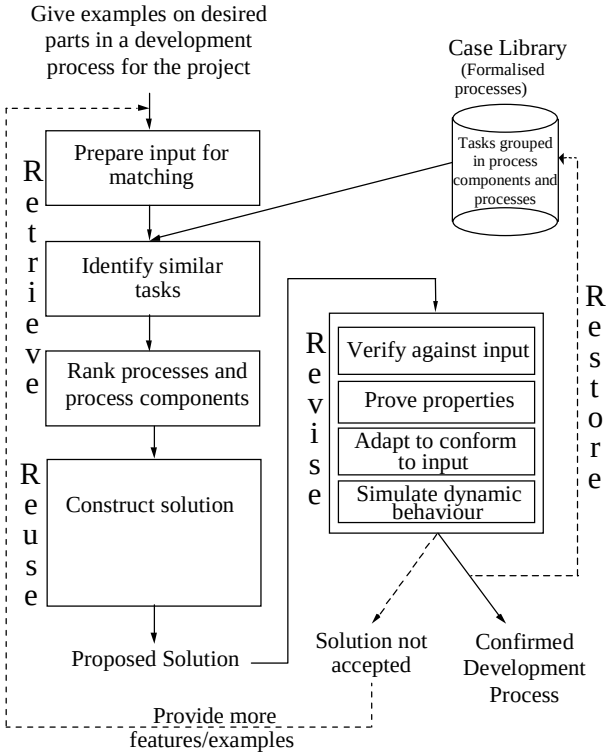

Figure 4: Outline of the CABS approach

If the solution is accepted (after verification against input examples and simulation of the behavior), the new or adapted system development process is stored in the case library but should not be available for reuse until the project has been successfully completed.

7. Conclusion

Using case-based reasoning in combination with formalized system development processes offers certain advantages over current practice in which system devel-

opment processes are mostly informal. The proposed CBR approach also offers advantages if used together with tools supporting the user in creating a specific system development process. The prime advantage is that successfully adapted system development processes are now available for reuse. Adapted system development processes which have been successfully used in a project are automatically made available for reuse (less successful processes may also be kept in the case library to avoid similar less successful processes in future, this is not implemented). This enables an organization to preserve locally gained experience in terms of improved and adapted system development processes.

In our further work we 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 users with specific projects for which they need to develop a system development process. The case-based reasoning prototype and graphical editor is currently adapted to fit the domain of development processes and a suitable graphical notation for system development processes will be chosen.

References

[1a] 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.

[1b] Doheny J.G. and Filby I.M. (1996) Modelling Software Development Processes and Standards, Technical Report AIAI-TR-205, University of Edinburgh, pp 1-10.

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

[3] 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.

[4] Zave P. and Jackson M. (1996). Four Dark Corners of Requirements Engineering. ACM pp 1-34.

[5] Aamodt A. (1993). A Case-Based Answer to Some Problems of Knowledge-Based Systems. Scandinavian Conference on Artificial Intelligence. E. Sandewall, C.G. Jansson (eds.), IOS Press, pp 168-182.

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

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

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

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

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