Fraunhofer Einrichtung
Experimentelles
Software Engineering

Modelling the Application Domains of Software Engineering Technologies

Technical Report

Author:
Andreas Birk

In part supported by
ESPRIT Project no. 23239 “PROFES”

IESE-Report No. 014.97/E
Version 1
August 14, 1997

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
IESE transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competetive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
D-67661 Kaiserslautern

Abstract

The effectiveness of software engineering technologies depends very much on the situation in which they are applied. In order to further improve software development practices we need to investigate and document the application domains of technologies. A first step toward a deeper understanding is to develop an appropriate modelling formalism that allows us to explicitly describe our current knowledge and to utilize the knowledge within software engineering research and practice.

This paper suggests a modelling formalism for describing application domains of software engineering technologies. It relies on established techniques from software domain analysis and software engineering measurement. A lifecycle process for acquisition and use of technology domain models is presented. The models can be used for supporting systematic reuse of technologies during planning of software projects and improvement programmes.

Keywords Domain Modelling, Knowledge Acquisition, Reuse

1 Introduction

Software management wants to make informed decisions. It seeks for decision support to identify the technologies that meet best the goals and characteristics of a software project or improvement programme.

A scenario of informed project planning could look as follows: The manager defines the goal of the project, e.g., to produce highly reliable software, and describes the known characteristics of the project (e.g., average time constraints, highly experienced engineers, object-oriented development, control software, etc.). Then, he/she enters these characteristics into a software-engineering information system and receives a recommended technology through which the goal can be achieved in the given context situation.

To date, this scenario is a vision. Neither do we have comprehensive repositories of best practice that are structured according to relevant goals and context characteristics, nor have we much documented experience about the context dependencies of software engineering technologies that would allow us to construct such repositories from scratch.

Aim of the work reported in this paper is to support the reuse of software engineering technologies through explicit models of their application domains. The targeted result is a repository of best practice that packages technologies together with information about when they should be used.

Strategy of our work is to (1) assess the dependencies between technologies and their application context, (2) develop a modelling schema for describing technology application domains, (3) describe the lifecycle of technology domain models (i.e., their construction through knowledge acquisition and their use within decision support for project planning), and (5) use these techniques for developing domain models of selected technologies (e.g., software verification techniques). In addition we develop software tools for the automated support of technology domain analysis and decision support. The results are evaluated empirically in industrial environments.

This paper presents a modelling schema for defining domain models of software engineering technologies. It relies on established techniques from software domain analysis and software engineering measurement. The lifecycle of technology domain models is described. It covers domain analysis through knowledge acquisition and decision support during planning of software projects and improvement programmes.

Our assumption is that many of today's software quality and improvement needs can be satisfied through more focused and better informed application of software engineering technologies. Every technology has its specific contexts where it can be applied successfully. In other situations other technologies are more appropriate. In addition, we assume that there exists much knowledge about the application domains of software engineering technologies in the minds of experienced software professionals. But this knowledge has seldom been acquired and documented explicitly. Hence, it can hardly be accessed and reused.

The work presented here is being conducted at the Fraunhofer Institute for Experimental Software Engineering (Fraunhofer IESE). We have gained experience with the assessment of software engineering technologies in numerous industrial measurement programmes. This work is directly related to ESPRIT project Profes[1] that develops a method for improving software processes based on customer-driven product quality factors in the embedded systems domain. Therefore, the relationships between process parameters (e.g., technologies), the project context, and product quality factors are investigated at three industrial sites.

The structure of the paper is as follows: Sections 2 to 3 briefly introduce the basic concepts underlying this work, namely software engineering technologies, domain analysis, and software engineering measurement. Core of this paper is Section 5 that presents a knowledge representation scheme for modelling application domains of software engineering technologies. The lifecycle of technology domain models is outlined in Section 6. Finally, the results are discussed and conclusions are drawn (Sections 7 and 8).

1 ESPRIT project no. 23239 "Product-focused improvement of embedded software processes" (cf. http:/www.iese.fhg.de/Profes)

2 Software Engineering Technologies

Throughout this paper, *software engineering technology* is used as generic term for denoting techniques, methods, and tools. A *technique* is a basic algorithm or set of steps to be followed in constructing or assessing software (cf. [BCR94a]). For instance, code reading by stepwise abstraction is a technique for assessing code.

A *method* is an organized approach based upon applying some technique. A method has associated with it a technique, as well as a set of guidelines about how and when to apply the technique, when to stop applying it, when the technique is appropriate and how we can evaluate it. For instance, a method will have associated with it a set of entry and exit criteria and a set of management supports. Code inspection is a method based upon some reading technique, which has a well-defined set of entry and exit criteria as well as a set of management functions defined for how to manipulate the technique (cf. [BCR94a]). A *tool* is a computer-based implementation of a technique or method.

Software engineering technologies are used for constructing or analysing software. During project planning, a software development process is designed by integrating several techniques, methods, and tools (cf. [BCR94a]).

As example technology throughout this document, we focus on verification technologies such as testing and reading techniques. These technologies have achieved a high maturity but there is still need for further investigation of their application domains.

3 Domain Analysis

Domain analysis is the process of identifying and organizing knowledge about some class of problems (i.e., the problem domain) to support the description and solution of those problems [AP91]. Typically, domain analysis supports the reuse of software architectures and components when specifying and implementing software systems. It has been argued that also other artifacts such as processes, technologies, and prediction models should be subject to reuse ("comprehensive reuse") [BR91].

Software domain analysis is the first step within the domain engineering process, prior to the specification and implementation of a reuse infrastructure [Ara89]. It produces a *model of the domain* that allows to specify, talk about, and model systems in that domain. Domain models structure repositories of reusable artifacts in order to support their identification and retrieval.

A common technique for domain modelling is faceted classification (cf. [PF87] [Pri87]). It originally stems from library science and is used there for describing and categorizing literature titles. An entity is characterized through multiple facets. Each facet contains a list or taxonomy of terms that describe its possible occurrences. Table 1 shows an example faceted classification scheme. A reusable component is characterized as tuple over the facets, e.g., (Programming language: C, Amount of reuse: less than 40%, Experience of developers: average, Size of project: small (one-site)).

programming language	amount of reuse	experience of developers	size of project
Fortran	less than 10%	low	**small (one-site)**
Ada	**less that 40%**	**average**	large (one-site)
C	less than 70%	high	multi-site
C++	70% or more		
Java			
Smalltalk			

Table 1 Example faceted classification scheme for classifying code verification technologies.

Technology domain analysis is widely analogous to software domain analysis (see Table 2) such that techniques and principles from software domain analysis can be transferred and adopted. Faceted classification is useful for describing the application domains of software engineering technologies. It allows for an intuitive characterization of technology application contexts.

Unlike the description of software components the modelling of technology domains usually involves abstract terms for which no generally shared definition exist. Examples are “average experience of developers” and “large project team”. In order to be operational, the faceted classification scheme should be complemented by explicit definitions of the involved terms. A useful technique for operational definition of abstract software engineering concepts can be adopted from software engineering measurement. It is introduced in the following section.

Software domain analysis concepts	Technology domain analysis concepts
Executable software system	Enactable software engineering process
Software component	Software engineering technology
Software architecture	Lifecycle methodology
Software architecture element	Lifecycle phase
Domain = Environment in which the software system is executed (e.g., business process, technical system)	Domain = Environment in which the software engineering process is enacted (e.g., software project, software organization)

Table 2 Analogies between software domain analysis and technology domain analysis.

4 Software Engineering Measurement

Software engineering measurement is a technique or method that applies software measures to software engineering objects (e.g., code, design documents, development activities, etc.) to achieve a predefined goal. Software measures assign objects in the software engineering world to objects in a formal system (i.e., numbers or symbols). Typical goals of software engineering measurement are to better understand software, to manage software processes, or to guide process improvement (cf. [BCR94c] [SEL94]).

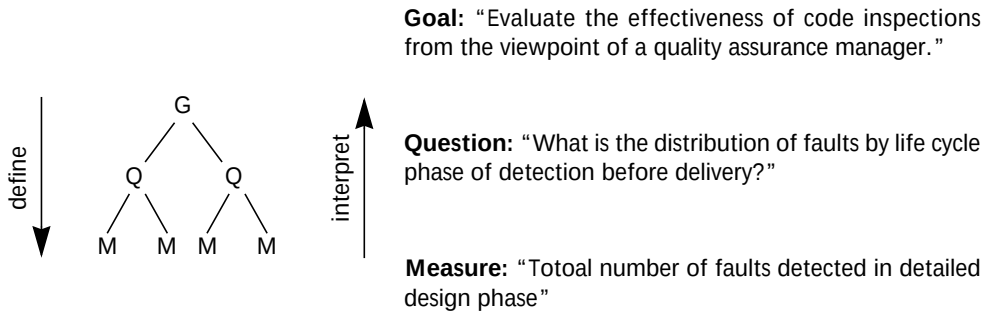

Figure 1 The strategy of GQM-based software engineering measurement,and an example goal, question, and measure.

A common approach to software engineering measurement is Goal/Question/Metric (GQM) [BCR94b] [GHW95] [BDR96]. It defines principles, documents, and processes for goal-oriented measurement in software engineering. GQM-based measurement starts with identifying a goal of measurement (see Figure 1). The goal is then defined operationally in terms of software measures. This is done through a process of hierarchical decomposition via the identification of questions that ask for the information needed to achieve the goal. The decomposition is determined by a person who shares the view from which the goal is defined (e.g., quality assurance manager). Figure 1 depicts the decomposition schematically and shows example goal, question, and measure.

For each measure a data collection procedure is defined. The collected data is interpreted in order to achieve the goal, following the inverse of the decomposition structure. The GQM approach assures that information about abstract aspects of software engineering can accurately be derived from empirical data in a context- and view-specific manner.

The unambiguous decomposition of abstract software engineering concepts into operational measures as it is provided through the GQM approach is a solution to the problem of lacking definitions for intuitive terms that can appear in faceted classification schemes for technology application domains (see Section 3). For that reason, the knowledge representation schema for technology domain models that is presented in the next section integrates principles from faceted classification with principles from GQM.

5 Modelling Technology Application Domains

This section introduces a modelling scheme for defining application domains of software engineering technologies. In the following, we refer to this scheme as *technology domain model* or just *domain model*. A technology domain model must accurately reflect the nature of software engineering technologies and their usage. It must also be appropriate for its later use within decision support. Therefore, we first describe the conceptual basis of and requirements on technology domain models. Then the modelling scheme is presented.

5.1 Conceptual Basis

Our aim is to describe the class of context situations (e.g., kinds of software projects) in which a software engineering technology can be applied successfully. Successful application of a technology (e.g., reading by stepwise abstraction) means that using the technology for a particular task (e.g., code verification) leads to achieving a desired goal (e.g., highly reliable software). Whether a technology can be applied successfully depends on the characteristics of the context situation (e.g., programming language, amount of reuse, experience of developers, etc.). This conceptual model is depicted in Figure 2.

Given a technology, its task, and its goal, we want to model the context characteristics under which the technology can be applied successfully. This set of context characteristics describes the domain of the technology. A domain comprises all concrete situations that share the defined characteristics.

A domain model depends on the viewpoint from which it has been derived. For instance, experienced software developers will describe a domain differently than project managers. Without further validation, a domain model is only valid in the overall environment in which it has been derived. Usually, the environment of a domain model is the software development organization whose projects are classified by the model.

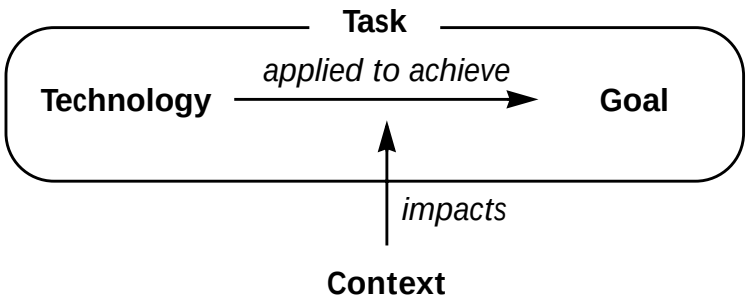

Figure 2 Conceptual model of technology application underlying domain analysis for software engineering technologies.

5.2 Requirements

Based on the conceptual model and the assumptions presented in the previous sections, the following requirements on a knowledge representation scheme for domain models can be identified:

- A domain model should describe an application domain in intuitive terms that meet the language of project planners who use it and experienced software professionals from whom it is acquired.
- A domain model should describe an application domain in an unambiguously and operationally, such that every real situation can be classified in terms of the domain model.
- A domain model should reference the technology, its application field, its goal, and the overall environment for which the domain model has been derived.
- A domain model should be marked as being specific to the viewpoint and the environment from which it has been derived.

At a first glance, the requirements on intuitive and unambiguous representation of context characteristics seem to contradict each other. The knowledge representation scheme presented in the following section is designed such that both requirements are fulfilled.

5.3 Domain Models

This section introduces a knowledge representation scheme for defining technology application domains. It fulfils the requirements imposed in the previous section.

Overall structure

A technology domain model consists of four parts (see Figure 3): header, intuitive domain model, operationalisation, and operational domain model.

The header relates the domain model to its technology, task, goal, viewpoint, and environment. The intuitive domain model is a faceted classification scheme that describes the domain using terms from the language of software professionals. The operationalisation translates the intuitive domain model into an operational domain model following the principles of GQM.

Header

A header contains five parts: technology, task, goal, viewpoint, and environment.

For every repository of domain models, a taxonomy of allowed contents should be defined for each part of the header. Figure 4 shows some example taxonomies. A goal can be stated as combination of a type of software product and a product quality.

Intuitive domain model

An intuitive domain model has the form of a faceted classification scheme. Each facet is represented by a domain factor such as "Programming language" or "Size of project". Each domain factor is defined through a set of discrete values ("possible values"). A facet of a concrete domain is characterized through one or more values ("actual values") from the set of possible values. A domain factor together with its actual value(s) is called domain characteristic.

Each domain factor has associated with it a concise definition in natural language that expresses the semantics of the factor. For instance, "Size of project team" can be defined as: "Size of the project team in (a) average number of person months over the entire duration of the project and (b) structure of the project team (one-site vs. multi-site). Project team involves all persons who are designing, implementing, and verifying the software product. Related hardware or marketing personnel who participate in requirements specification or validation activities are not considered."

Operationalisation

The operationalisation maps the intuitive domain factors to concrete domain factors that belong to the operational domain model (see below). There are three kinds of mappings:

- Direct operationalisation

- Functional operationalisation
- Vague operationalisation

Direct operationalisation is to be used for intuitive domain factors that can already be determined unambiguously for every real context situation. An example is “Programming language”.

Functional operationalisation is to be used for intuitive domain factors that should be decomposed in one or more elementary domain factors. The relationship between the elementary factors and the intuitive ones must be defined in a numerical or logical function. An example is “Amount of reuse” which can be calculated from the total number of modules and the number of reused modules.

Vague operationalisation is to be used for intuitive domain factors that can be traced down to more elementary factors, but for which the relationship between the factors can not be defined precisely. Every possible value of such a domain factor should be defined as linguistic variable using a fuzzy set [Zad94]. An example is “Experience of developers”.

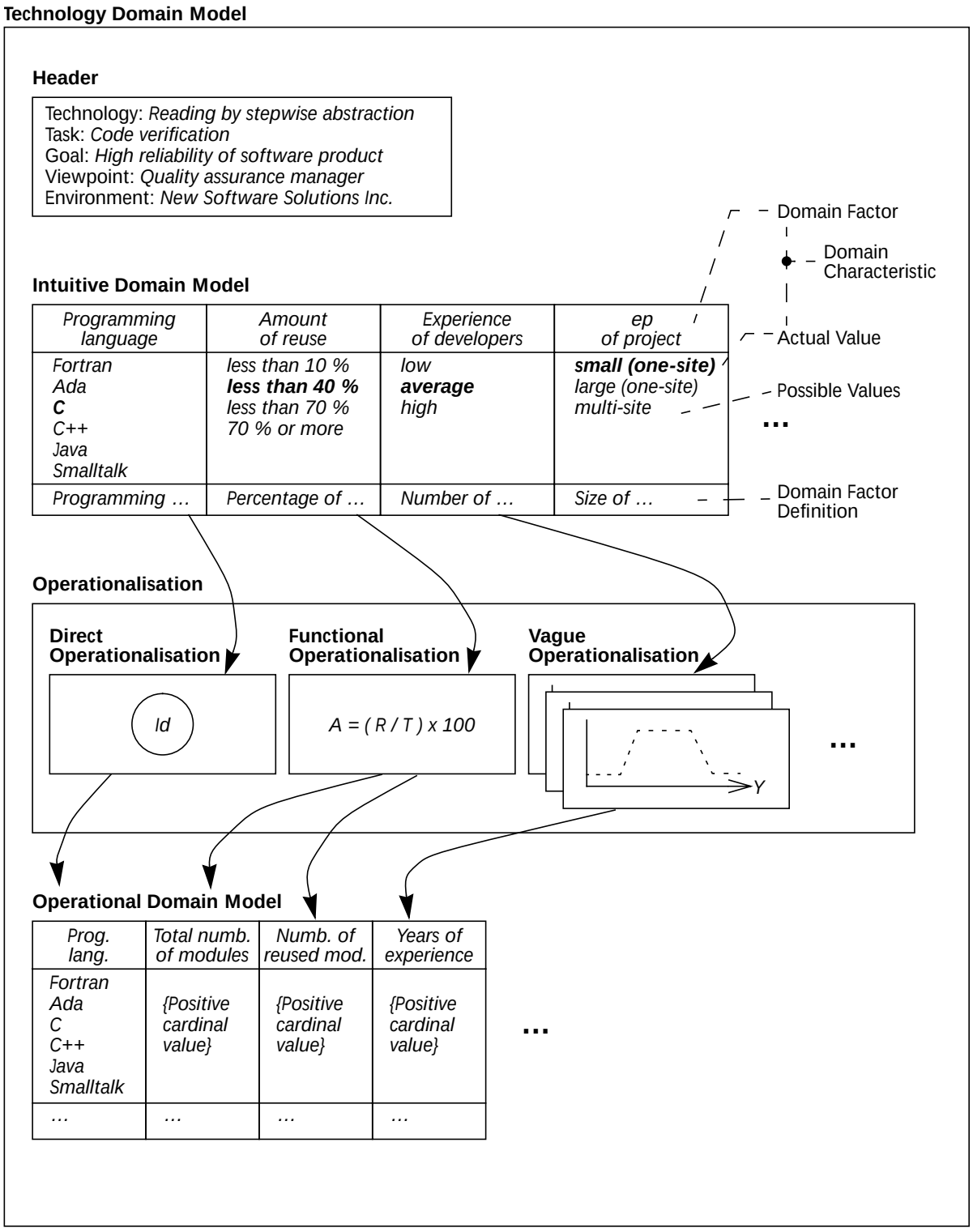

Figure 3 Structure of technology domain models.

Operational domain model

The operational domain model consists of a set of domain characteristics that with real-world phenomena which can be clearly be determined for every context situation. Usually, the domain factors of an operational domain model are more elementary than those of the intuitive domain model. The structure of both kinds of domain models is the same.

Tasks

Software lifecycle
Requirements engineering
Design
Coding
Verification
Validation
Project management
Configuration management
Software engineering measurement

Products

Software product
Software development process model
Problem description
Requirements description
System Design
Code
Project plan
Configuration plan
Measurement plan

Product Quality

Quality
- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Time
Cost

Viewpoints

Software development
Project management
Configuration management
Quality assurance

Figure 4 Example software engineering taxonomies that can be appropriate for use within technology domain models: Tasks (cf. [RV95]), products (cf. [RV95]) , product goals(cf. [ISO91]) , and viewpoints (cf. [RV95]).

6 Technology Domain Model Lifecycle

This section presents the targeted lifecycle of technology domain models. First, the construction of domain models is addressed briefly. Then, it is outlined how domain models can be used for systematically selecting and reusing technologies during the planning of software projects and improvement programmes.

Domain models can be derived through various strategies: experiments and quasi-experiments, case studies, software measurement programmes, subjective experience reports, theoretical analysis, knowledge acquisition, and others. We focus on knowledge acquisition backed up with theoretical analysis, because this is expected to be the most efficient way for achieving useful initial results. A knowledge acquistion process has been defined that involves open and structured interviews for operationalising and defining the intuitive domain models.

Knowledge acquisition involves the risk of leading to less accurate results than empirical method. We expect to avoid this risk by a systematic and theoretically well-grounded knowledge acquisition method, the conducting of multiple interviews with different experts, and finally the careful use of the domain models within a decision support method that integrates the discerning of a human decision maker.

Given a repository of domain models available, how can it be used for supporting technology selection during project planning? In order to be valuable for project planning, such a repository should contain domain models for multiple technologies that share the same task, goal, viewpoint, and environment. Figure 5 depicts the overall process of technology selection using domain models.

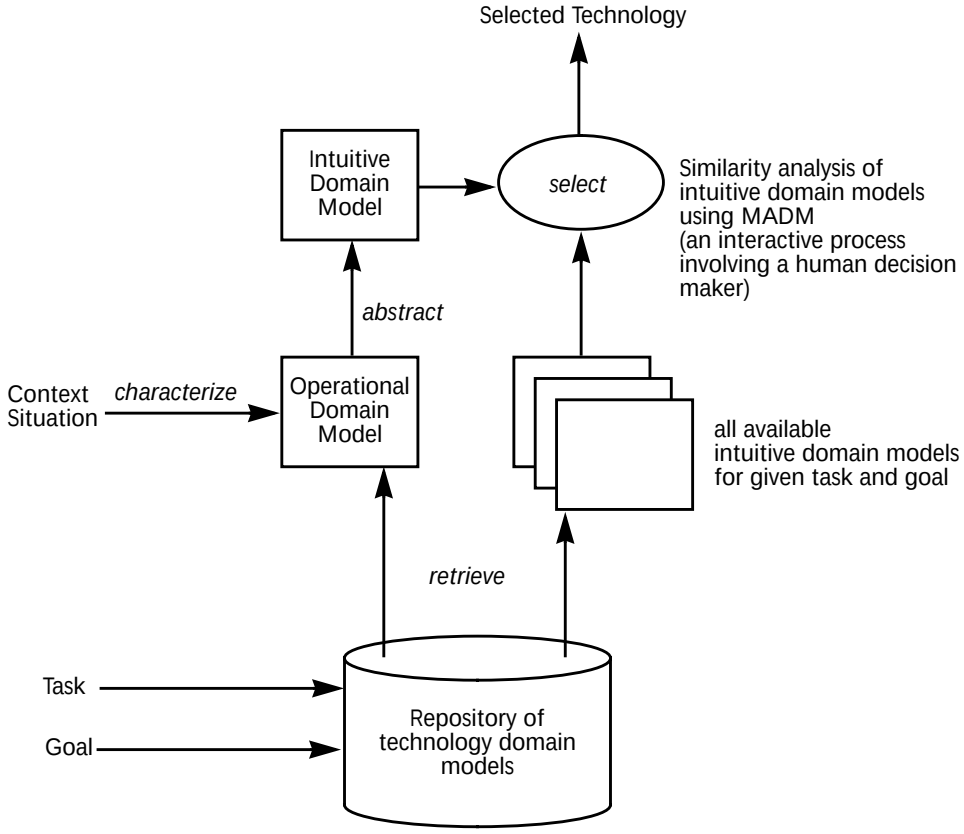

Figure 5 Systematic selection of software engineering technologies using domain models.

Initially, task and goal for which a suitable technology is searched are determined This allows to retrieve the associated operational domain model as well as all available intuitive domain models from the repository of domain models. The operational domain model is used to characterize the given context situation. The intuitive domain model is abstracted from this characterisations. The actual selection of the technology is performed based on intuitive domain models. The technology is selected whose intuitive domain model matches best the intuitive domain model of the current context situation. This is an interactive procedure, in which a human decision maker is involved and assisted by decision support techniques such as multiple-attribute decision methods (MADM) [MP97].

7 Discussion

Related work is currently performed at University of Nebraska-Lincoln by Scott Henninger and at University of Maryland by Carolyn Seaman, Victor Basili, and others. Henninger's approach aims at gaining experiences on technology usage and domains and formalizing the experiences gradually as knowledge grows and matures [HLR95] [Hen96]. He uses a case-based repository to collect experiences from software practitioners during the course of their daily work. These experiences can be used directly in order to complement existing manuals and handbooks. They also provide the basis for detailed elaboration of and abstractions from the recorded cases. Henninger emphasizes that experiences are domain-specific an that their formalization goes hand in hand with a better understanding and definition of domains.

At University of Maryland, focus is put on identifying and evaluating methods for eliciting domain information [BBT94] that utilize qualitative and quantitative techniques of empirical research. The work is being conducted in co-operation with NASA's Software Engineering Laboratory at Goddard Space Flight Centre aiming at identifying similarities between different software development units that allow for the systematic transfer of software engineering knowledge and experiences.

The presented schema for modelling application domains of software engineering technologies is a formalization of the object context of reusable artifacts (in our case technologies) that has been introduced by Basili and Rombach [BR91]. It supports the retrieval and reuse of technologies following the experience factory paradigm [BCR94a].

The technology domain modelling schema has been evaluated and shown its appropriateness using existing data from industrial software measurement programmes. Currently, the knowledge acquisition method for technology domain analysis is being refined and tool support for knowledge formalisation is being developed. It will then be evaluated in a case study with a industrial software organisation.

The decision support method is also being refined and a software prototype for supporting technology selection is being developed. It will be realized on top of a case-based reasoning system that provides a repository of domain models and functionality for similarity-based retrieval [AB+97]. The interactive procedure for technology selection will utilise selected multiple-attribute decision methods [MP97].

8 Conclusion

A modelling schema has been presented for describing the application domains of software engineering technologies. It is designed for being constructed through knowledge acquisition and used for technology selection during the planning of software projects and improvement programmes. The modelling schema has been successfully evaluated using existing data from software measurement programmes.

The lifecycle of domain models has been outlined, covering knowledge acquisition and decision support for technology selection and reuse. Knowledge acquisition is beneficial for uncovering still implicit knowledge of experienced software professionals. Support for technology selection and reuse leads to more informed decision making and planning in software engineering.

The schema for technology domain modelling is the first step toward automated decision support for technology selection and reuse. It guides the refinement and formalisation of technology selection and knowledge acquisition processes. Repositories and software tools can be designed that implement the models and processes.

The presented approach to technology domain modeling is expected to provide us with a repository of technology / context dependencies. This will deepen our understanding of success factors for technology application. Software engineering practice can benefit through the systematic selection of technologies for given tasks and goals. Software engineering research can gain a more comprehensive empirical basis for the focused development and tailoring of best practice technologies.

9 Acknowledgements

The author thanks Klaus-Dieter Althoff, Lionel Briand, Frank van Els, Markus Hoffmann, Oliver Laitenberger, Dietmar Pfahl, Günther Ruhe, and Carsten Tautz for their comments and the discussions on this paper and the reported work. This work has been partly supported by the CEC through ESPRIT project no. 23239, “Profes”.

10 References

[AB+97] Klaus Althoff, Andreas Birk, Christiane Gresse, Carsten Tautz. "CBR for experimental software engineering". To appear in: Brigitte Bartsch-Spörl et al. Case-Based Reasonig Technology from Foundations to Applications. Springer, Berlin, 1997. (in preparation).

[AP91] G. Arango and R. Prieto-Diaz. Domain analysis concepts and research directions. In G. Arango and R. Prieto-Diaz (Eds.), Domain Analysis and Software Systems Modeling (pp. 9–33). IEEE Computer Society, Washington, DC, 1991.

[Ara89] G. Arango. Domain analysis: From art form to engineering discipline. In Proc. of the Fifth International Workshop on Software Specification and Design (pp. 152–159). IEEE Computer Society, Washington, DC, 1989

[BBT94] V. Basili, L. Briand, and W. Thomas. Domain Analysis for the Reuse of Software Development Experiences. In Proc. of the 19th Annual Software Engineering Workshop, NASA/GSFC, Greenbelt, MD, December, 1994.

[BCR94a] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Experience Factory. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 469–476. John Wiley & Sons, 1994.

[BCR94b] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Goal Question Metric Paradigm. In John J. Marciniak, editor, Encyclopedia of Software Engineering, volume 1, pages 528–532. John Wiley & Sons, 1994.

[BCR94c] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Measurement. In John J. Marciniak, editor, Encyclopedia of Software Engineering, volume 1, pages 646–661. John Wiley & Sons, 1994.

[BDR96] Lionel C. Briand, Christiane M. Differding, and H. Dieter Rombach. Practical guidelines for measurement-based process improvement. Technical Report ISERN-96-05, Fraunhofer Institute for Experimental Software Engineering, Sauerwiesen 6, 67653 Kaiserslautern, Germany, 1996.

[BR91] Victor R. Basili and H. Dieter Rombach. Support for comprehensive reuse. Software Engineering Journal, 6(5):303-316, September,

1991.

[GHW95] Christiane Gresse, Barbara Hoisl, and Jürgen Wüst. A process model for GQM-based measurement. Technical Report STTI-95-04-E, Software Technologie Transfer Initiative Kaiserslautern, Fachbereich Informatik, Universität Kaiserslautern, D-67653 Kaiserslautern, 1995.

[Hen96] Henninger, "Accelerating the Successful Reuse of Problem Solving Knowledge Through the Domain Lifecycle," Fourth International Conference on Software Reuse, Orlando, FL, 1996 , pp. 124-133.

[HLR95] S. Henninger, K. Lappala, A. Raghavendran "An Organizational Learning Approach to Domain Analysis", Seventeenth International Conference on Software Engineering - ICSE-17, (Seattle, WA), 1995, pp. 95-104.

[ISO91] ISO/IEC. Information technology - Software product evaluation - Quality characteristics and guidelines for their use. ISO/IEC standard 9126. Geneva, Switzerland, 1991

[MP97] M. Molloghasemi, J. Pet-Edwards. Making multiple-objective decisions. IEEE Computer Society Press, Washington, DC, 1997.

[PF87] R. Prieto-Diaz and P. Freeman. Classifying software for reusability. IEEE Software, 4(1):6–16, January 1987.

[Pri87] R. Prieto-Diaz. Domain analysis for reusability. In Proc of COMPSAC 87: The Eleventh Annual International Computer Software & Applications Conference (pp. 23–29). IEEE Computer Society, Washington, DC, 1987.

[RV95] H. Dieter Rombach, Martin Verlage "Directions in Software Process Research" Advances in Computers, Volume 41, Marvin V. Zelkowitz (Ed.), Pages 1-63, Academic Press, Boston, MA, 1995.

[SEL94] National Aeronautics and Space Administration. Software measurement guidebook. Technical Report SEL-84-101, NASA Goddard Space Flight Center, Greenbelt MD 20771, July 1994.

[Zad94] L. Zadeh. Soft computing and fuzzy logic. IEEE Software, 11(6):48–56, November 1994.

Document Information

Title:	Modelling the Application Domains of Software Engineering Technologies
Date:	August 14, 1997
Report:	IESE-014.97/E
Version:	1
Status:	Final
Distribution:	Public