ISSN: 2321-7782 (Online)

Volume 3, Issue 7, July 2015

International Journal of Advance Research in Computer Science and Management Studies

Research Article / Survey Paper / Case Study

Available online at: www.ijarcsms.com

Software Engineering Framework for software Industry using Heterogeneous and Composite System

K.Veerasamy[1]
Asst. Professor
Department of Computer Applications,
Karpagam University, Echanari, Coimbatore- 641021

S. Punitha[2]
Asst. Professor
Department of Computer Applications,
Karpagam University, Echanari, Coimbatore- 641021

Abstract: Software engineering has made significant contributions to "engineering-in-the-large". The nature of the software process has been researched, and computer based tools and environments have been built to support this process. Other more established engineering disciplines, such as instrument design, have developed professional practices, mature mathematical frameworks for system modeling and accepted quality standards lacking in software engineering. Little effort however, has been devoted to the cross-fertilization of software engineering and engineering design, or indeed the exploitation of the frequently observed commonalities between them. The Software Engineering and Engineering Design (SEED) project described in this article has attempted to address these issues through the study of heterogeneous, composite systems. This has resulted in a model of the engineering design process, an organizational framework for systems development methodology and integrated computer-based support for this framework.

I. INTRODUCTION

Many large and complex systems deploy a variety of different technologies, and require a variety of development strategies and notations to specify their behavior. Modern instruments for example, have substantial software components alongside their electronic and mechanical hardware. Such systems require the coexistence, even the incorporation of, software engineering methods within the traditional engineering design process. There are clear similarities between the disciplines of software engineering and instrument design, yet enough differences to tempt the transfer of successful development techniques from either discipline to the other. The SEED project [Fink90] has systematically studied these similarities and differences in its attempt to transfer technology and expertise from one discipline to the other. Concurrently, an organizational framework for systems development methodologies has been constructed to describe, manage and apply the engineering design process to the development of heterogeneous, composite systems [Fink92].

SEED is a collaborative project between Imperial College and City University, and builds on the substantial experience of the partners in supporting software development and instrument design. Software Engineering. While computer scientists devise improved techniques for structuring and programming large, complex systems, software engineering research focuses on the controlled management of such techniques within the context of the software development "life cycle". As software systems have grown in size and complexity, software engineering as a discipline has focused on the process of software development.

Software development projects encompass a range of activities that precede, include and follow programming. Foremost among these activities is the elicitation, specification and analysis of system requirements. Requirements specification is now recognized as the essential first step in any systems development process, and its documentation is often the contractual reference against which system designs are validated. Software engineering research has produced a multitude of specification and design methods that may be used to describe system requirements and design architectures. These methods typically utilize a number of different representation styles or notations together with prescriptions of how to go about producing specifications using these notations. Many general problems such as incompleteness, inconsistency and ambiguity in specification have been encountered, and powerful approaches developed to try and resolve them.

Computer aided software engineering (CASE) tools and integrated programming support environments (IPSEs) have emerged to provide practical, automated support for these methods. Such computer- based tools provide a means for enacting methods' underlying process models using the notations prescribed by these methods. Considerable experience has now been gained within the software engineering community in both CASE tool technology and the underlying development methods which CASE tools support.

Instrument Design. Instruments are an interesting class of engineering artifacts. They are composite systems, consisting of a large number of interacting sub-components and employing a variety of different technologies (mechanical, electrical, information processing, even biological and chemical). They are a class of artifacts whose general properties are well known and in the design of which there is considerable expertise. Instrument systems are therefore an excellent vehicle for exploring heterogeneous systems development and inter-disciplinary technology transfer.

Technology Transfer. Technology transfer deals with the problems of fitting technology into a new setting. While this transfer is commonly perceived as flowing from research into industry, the SEED project has concentrated on the inter-disciplinary transfer of technology between software engineering and engineering design. The aim of the project has therefore been to apply the techniques, methods and tools deployed by one discipline to solve problems of another.

Both CORE and Z deploy a systematic process and use rich representations to produce descriptions of function and behavior. This was reflected in the system specifications produced by the two methods. In both case studies, the use of software specification methods produced clear and concise specifications of the function and behavior of the engineering artifacts. Other areas of transfer continue to be investigated. In particular, the specification of so called "non-functional requirements" that deals with aspects of systems that are difficult to quantify remains problematic. The authors' approach has been to attempt to quantify and formalize non-functional requirements, so that they may be expressed and analyzed as functional ones.

This is in line with current trends in software engineering where, for example, researchers in Human-Computer Interaction (HCI) have sought to formalize definitions of user interface properties to provide a consistent "look" and "feel" to interfaces.

II. The Engineering Design Process

The formulation of a satisfactory model of the design process is a fundamental concern of many disciplines. It is treated, in particular, in the literature of engineering design, systems science, planning, and creativity. The motivation for the concern with the topic is the provision of a conceptual framework for the organization of design activity, the support of the creative work of designers, the effective teaching of design and, finally, the automation of (or automated support for) design.

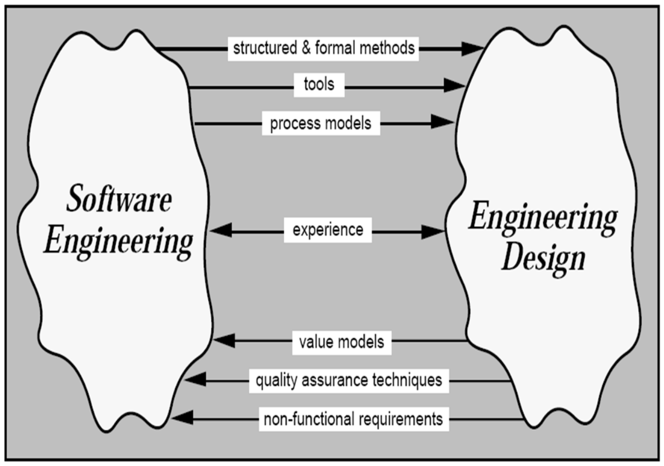

structured & formal methods
tools
process models
Software Engineering
experience
Engineering Design
value models
quality assurance techniques
non-functional requirements

Figure-1: Some areas of technology transfer between software engineering and engineering design. Arrows show the direction of transfer.

There exists an extensive literature of the topic. The authors have, among others, reviewed the literature of the classical views of design methodology and presented the generally accepted model of the design process [Fink83]. More recently Burton [Burton90] has reviewed the literature comprehensively and analyzed critically the evidence in support of the generally accepted, or consensus, model. As part of the SEED project, the deficiencies of the classical model have been examined.

Classical Consensus Model. While there are significant differences in the many presentations of models of the design process, they all fall within a common abstract model. The model is partly *descriptive*, an attempt to give an account how design is actually carried out, and partly *prescriptive*, a recipe how design should be carried out. The latter is sometimes the result of systematic external observation of design activity and more often on participant observation, which is often indistinguishable from the theoretical, introspective rationalization of the experienced designers. The essence of the consensus model, as it is seen by the authors, is as follows.

Design is considered as a complex information system, which transforms the statement of the perception of a want and the commitment to satisfy it, into a specification of a system or artifact to satisfy that want, such that the system or artifact can be made or implemented. The design process is built up of a sequence of elementary stages. The model of an elementary design stage is shown in Figure-2. Each stage is a sequence of processes: task definition, solution generation, solution analysis, solution evaluation and decision.

What is termed the task definition by the authors, is a process which transforms a model of the solution from the preceding stage of the design sequence into a requirement specification, including a value model for the solution of that stage. The requirement specification is passed on to a solution generation process, which produces a model of a candidate solution, which may satisfy the requirement specification.

The decision process receives the information about the value of the candidate solution and either accepts it as a specification of the solution to be used as the basis of the succeeding stage of design or else it either, returns to the generation of an alternative solution or, if the alternatives have been exhausted, it returns to modify the requirement specification. It may, if neither of the latter actions led to an acceptable solution return to the beginning of the preceding stage of the design process.

The model of the design process presented above is in accord with generally accepted accounts of the process, although the various models in the literature differ in detail in accordance with the perspective of the author, the domain of design considered, the size of system and the nature of the technologies involved. The model presented above describes what is widely considered to be "good design practice".

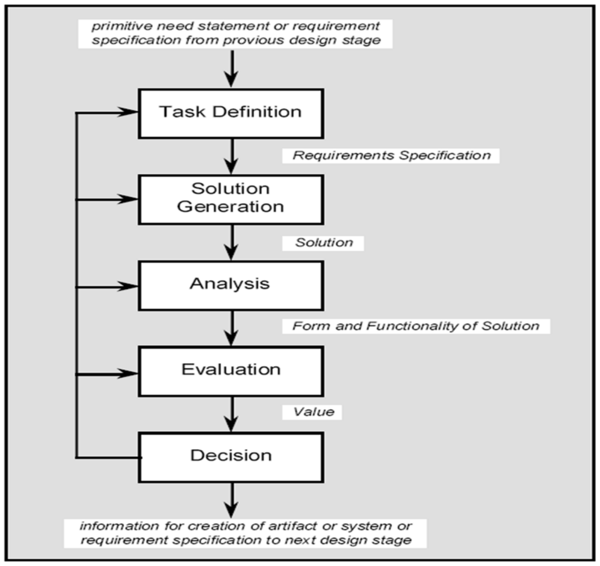

primitive need statement or requirement specification from provious design stage
Task Definition
Requirements Specification
Solution Generation
Solution
Analysis
Form and Functionality of Solution
Evaluation
Value
Decision
information for creation of artifact or system or requirement specification to next design stage

Figure-2: An elementary design stage in the classical consensus model of the engineering design process.

The model of the design process presented above is in accord with generally accepted accounts of the process, although the various models in the literature differ in detail in accordance with the perspective of the author, the domain of design considered, the size of system and the nature of the technologies involved. The model presented above describes what is widely considered to be "good design practice". It is based on a design process that moves from the establishment of design requirements to the generation of a solution. It recognizes that the sequence of requirement specification, candidate solution generation, analysis and evaluation is a logical necessity. It further recognizes that design is necessarily a succession of stages, say, planning, conceptual design, embodiment design and detailed design, which must be carried out in an orderly sequence.

Deficiencies in Classical Model. The model only considers top-down design, starting from a requirement, formulating a solution in terms of high-level components which can satisfy those requirements and moving downwards. However, bottom-up design, in which design solutions for components of the design are starting points, may on occasion be appropriate, since top-down design may lead to difficult component problems. Further, effective reuse of preexisting designs seems to be difficult to accommodate in a top-down design approach. Middle-out design may also be appropriate. Neither bottom-up nor middle-out approaches fit well with the classical model.

The model does not explicitly recognize that a number of candidate concepts may be generated and developed in parallel at any stage. Indeed decision involves in general the choice among a set of candidates. Nor does the model explicitly show the important place of partial solutions arrived at during the process, the processing of which may which may be abandoned at some point, but returned to at a latter point. Finally, the model does not explicitly show the place of knowledge in the design process, thereby rendering it largely unsuitable for assisting the developer; e.g., through automated design support. In order to remedy these deficiencies, and to provide an improved basis for work on design automation, the authors propose an integrated object-based framework which resolves the deficiencies of the consensus model and supports the design and construction of heterogeneous, composite systems.

III. An Integrated Framework

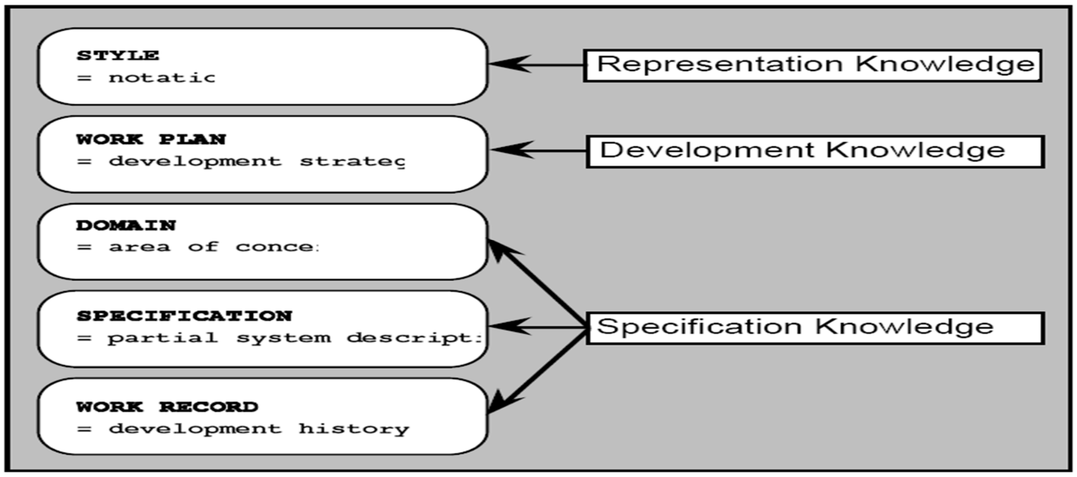
So far, many of the *differences* in the development of software and hardware systems have been highlighted, and attempts have been made to transfer successful techniques across disciplines. Meanwhile, the underlying model of any engineering development process has been described, highlighting the *similarities* between engineering disciplines.

Nevertheless, while technology transfer and a unified design approach greatly enhance the process of systems development, the development of heterogeneous, composite systems invariably requires heterogeneous approaches to their design. The ViewPoint Oriented Systems Engineering (VOSE) framework [Fink92] is an organizational framework that acknowledges this requirement. The framework supports multiple notations and development strategies to describe multiple components of composite systems. ViewPoints represent “agents” having “roles-in” and “views-of” a problem domain. Each ViewPoint describes a partial specification of the problem domain, presented in a particular notation and developed using a particular strategy.

Motivation. Design of engineering systems is a complex activity. To support people engaged in it, "methods" which guide and organize the activity are required. Such methods consist of the following components: a set of representation schemes, that is, ways of describing the system under design; a model of the design process and a means for using that model to generate guidance on what to do in particular circumstances. Such methods have a variety of uses: they can be used to guide individual designers; they can be used for management control; they can be used to set development standards and prescribe design deliverables; they can be used to give a development rationale; they can be used as a basis for principled tool support.

If we examine how knowledge is applied in design we can distinguish three classes of knowledge: development knowledge, knowledge about the process of design; representation knowledge, knowledge about how the artifact or system is to be represented; design knowledge, knowledge about the artifact or system itself and the domain or context in which it is to be placed that arises out of the design process. For the most part these three classes of knowledge have been treated separately: development knowledge through the study of models of the design process; representation knowledge, through the study of modeling techniques and specification languages; design knowledge through the study of design databases and CAD tools. Our framework attempts to tie these classes of knowledge together to construct methods.

ViewPoints. A ViewPoint may be defined as a loosely coupled, locally managed, coarse-grained object, encapsulating the representation knowledge, development process knowledge and design (specification) knowledge of a particular problem domain. This knowledge is described in the five "slots" shown schematically in Figure-3. The development participant associated with any particular ViewPoint is known as the ViewPoint "owner". The owner is responsible for developing a ViewPoint *specification* using the notation defined by in the *style* slot, following the strategy defined by the *work plan*, for a particular problem *domain*. A development history is maintained in the *work record*.



Methods. Many ViewPoints may employ the same development technique (e.g., top-down functional decomposition) to produce different specifications for different domains. We therefore define a reusable *ViewPoint Template* in which only the style and work plan slots are elaborated. A single ViewPoint template may then be *instantiated* more than once to yield different ViewPoints.

In general, a method is composed of a number of different development techniques. Each technique has its own notation and rules about when and how to use that notation. Thus, in the context of the ViewPoints framework, a *method* is a configuration (structured collection) of ViewPoint templates, the templates corresponding to the method's constituent development techniques.

Developments. A development is a configuration of ViewPoints instantiated from a method's ViewPoint templates. These ViewPoints are related via inter-ViewPoint consistency rules, that may be enacted when full (or partial) consistency is required. Each ViewPoint is locally managed, responsible for its own in- and inter-ViewPoint consistency, and potentially distributable both logically and/or physically.

Consider for example the development of a computer-based instrument system such as a digital storage oscilloscope (DSO). It is a heterogeneous system composed of electronic and information processing components. We may choose to develop the specification of this oscilloscope using system block diagrams, functional decompositions, data flow diagrams and structured text. Our method in this context is a set of four ViewPoint templates, a template for each of the above four development techniques. Our DSO development project is a configuration of ViewPoints, instantiated from the templates provided. We may thus have a single ViewPoint whose specification contains the overall DSO system block diagram, a number of ViewPoints whose specifications contain functional decompositions of the various blocks of the DSO, a number of

ViewPoints whose specifications contain data flow diagrams of the various components of the DSO, and several ViewPoints whose specifications contain structured text descriptions of various other ViewPoint specifications.

Integration. Integration is central to the ViewPoints framework. The framework may be used by method designers to integrate different development techniques, to build new methods, or simply to customize standard methods to their individual requirements. This is done by defining methods' constituent templates and the consistency relationships between them. Tool integration is treated as a special case of the more general method integration problem, and is thus a natural consequence of the method integration mechanisms of the framework. Individual tools may be constructed by tool developers to support individual templates, which are then integrated by the same inter-ViewPoint rules defined in the templates.

A development project in VOSE is a configuration of ViewPoints. These ViewPoints may be grouped together by common domain, template and/or arbitrary logical or managerial configurations. Whatever structuring mechanism is chosen, an appropriate management mechanism is needed to organize and navigate through large ViewPoint structures.

Tool Support. A prototype computer-based environment has been constructed to support the VOSE framework, and several sample tools supporting individual ViewPoint templates have been integrated into this environment [Nuseibeh92]. The environment, called *The*Viewer, was developed in bjectworks/Smalltalk and operates under X-windows, Macintosh OS or Windows for the PC. *The*Viewer provides support for method designers and method users (Figure-4), and facilitates ViewPoint template description, ViewPoint development and ViewPoint management (Figure-5, 6, and 7 respectively).

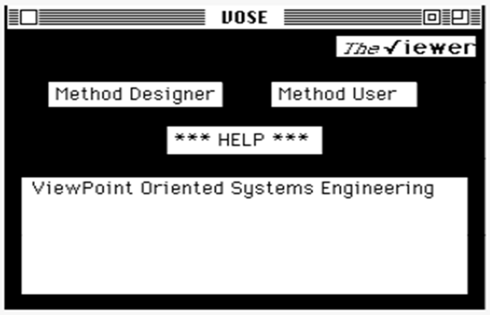


Figure-4: *The startup window of The√iewer. It defines the scope of the VOSE environment. The "Method Designer" button creates a Template Browser (Figure-5), while the "Method User" button creates a ViewPoint Configuration Browser (Figure-7).*

IV. Conclusions And Further Work

This paper has described a variety of issues surrounding inter-disciplinary technology transfer tackled by the SEED project. At one level, the authors have examined the differences between the disciplines of software engineering and engineering design and attempted to implant techniques from one into the other. At another level, the apparent similarities between the two engineering disciplines have been recognized and a unified model of the engineering design process has been constructed. This model fits into the proposed ViewPoint Oriented Systems Engineering (VOSE) framework, which acknowledges the similarities and differences between systems development disciplines, and attempts to provide both a framework and a mechanism for their integration. This has proved particularly relevant for the specification, design and construction of heterogeneous, composite systems.

The SEED project represents work still in progress. A sample tools have been integrated into this environment. The intention is to upgrade *The*Viewer from prototype status into a fully operational environment supporting the distributed development of heterogeneous, composite systems. Further work is still needed however in the area of consistency checking between different representations, and the mechanisms for their enactment and implementation. Modeling the ViewPoint oriented development process is also being investigated, with the objective of providing automated, computer-based guidance for the ViewPoint developer.

References

1. [Burton90] P.J. Burton, “A Model of the Design Process”, Ph.D. thesis, The City University, London, 1999.
2. [Fink83] L. Finkelstein and A. Finkelstein, “Review of Design Methodology”, IEE Proceedings, Volume 130, Pt. A, Number 4, pp.213-222, June 2003.
3. [Fink90] A. Finkelstein, T. Maibaum and L. Finkelstein, “Engineering-in-the-Large: Software Engineering and Instrumentation”, Proceedings of UK IT 2010 Conference (IEE), Southampton, Conference Publication 316, pp.1-7, 19th-22nd March 2008.
4. [Fink91a] L. Finkelstein, J. Huang, A. Finkelstein and B. Nuseibeh, “Using Software Specification Methods for Measurement Instrument Systems, Part 1: Structured Methods”, Measurement Journal, Vol. 10, No. 2, pp.79-86, Apr-Jun 2011.
5. [Fink91b] L. Finkelstein, J. Huang, A. Finkelstein, B. Nuseibeh, “Using Software Specification Methods for Measurement Instrument Systems, Part 2: Formal Methods”, Measurement Journal, Vol. 10, No. 2, pp.87-92, Apr-Jun 1992.
6. [Fink92] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein and M. Goedicke, “Viewpoints: A Framework for Integrating Multiple Perspectives in Systems Development”, International Journal of Software Engineering and Knowledge Engineering, Special issue on “Trends and Future Research Directions in SEE”, World Scientific Publishing Company Ltd., 1992.