DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCES
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529

A SOFTWARE DEVELOPMENT AND EVOLUTION MODEL BASED ON DECISION-MAKING

By

J. Christian Wild, Principal Investigator

Final Report
For the period ended January 31, 1991

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Under
Research Grant NAG-1-966
Dr. Dave E. Eckhardt, Jr., Technical Monitor
ISD-Systems Architecture Branch

Submitted by the
Old Dominion University Research Foundation
P.O. Box 6369
Norfolk, Virginia 23508-0369

June 1991

ACKNOWLEDGMENTS

The attached paper has been accepted for publication in the Third International Conference on Software and Knowledge Engineering to be held in Skokie, Illinois, June 1991. The paper is being submitted in lieu of a final report for the research project entitled, "A Project-Knowledge Base for Supporting Software Generation Based on Documenting Decision Dependencies," supported by the National Aeronautics and Space Administration, research grant NAG-1-966, Dr. Dave E. Eckhardt, Jr., of Information Systems Division, Systems Architecture Branch was technical monitor.

A Software Development and Evolution Model Based on Decision-making

Jinghuan Dong Chris Wild Kurt Maly
Department of Computer Science
Old Dominion University
Norfolk, VA 23529

Abstract

Design is a complex activity whose purpose is to construct an artifact which satisfies a set of constraints and requirements. However the design process is not well understood. In this paper we focus on the software design and evolution process and propose a three dimensional software development space organized around a decision-making paradigm. We present a initial instantiation of this model called $3DPM_p$ which has been partly implemented. Discussion of the use of this model in software reuse and process management is given.

1 Introduction

Design is a complex activity whose purpose is to construct an artifact which satisfies a set of constraints and requirements. Some of the characteristics of the design activity which distinguish it among problem solving activities are complexity and diversity of the design space, interdependencies between artifact structures, lack of well-defined criteria for evaluating resultant designs [6] and inability to articulate a complete and consistent set of requirements [15]. The intelligence and knowledge of the designers decide the success and failure of a design. In this paper we concentrate on software design which is further characterized by continued evolution of the initial design through a process commonly referred to as software maintenance.

The starting point for any design activity is a description of the problem given in the form of a set of requirements. In many design situations a requirement may be naturally stated in a sufficiently precise or formal manner that one can construct definite and applicable procedures for determining whether or not the design meets such requirements. Such problems are termed well-structured problem. However, many design problems are ill-structured in that there may be no definite, objective criterion or procedure for determining whether or not a design meets its requirements. This is the preciseness problem [4]. Other problems are inconsistency, incompleteness and ambiguity etc. Given the requirements, the designer may follow some kind of design process (model) or design strategy to analyze the requirements, to propose a candidate solution, to implement it and to test or verify it. However, the nature of the design process is not well understood. It was over 20 years ago since the "software crisis" was advanced. Some well-known methodologies have been proposed such as top-down analysis, object-oriented method. These methodologies greatly improve the productivity of software, but they haven't solved the "crisis" yet. One major problem is that most of these methods focus on product development. Recently, the investigation of software design process has received much interest[2]. This research claims that not only product information should be recorded, but also design process information pertaining to products should be stored in order to increase maintainability.

Here we define the software process as the collection of related activities, seen as a coherent process subject to reasoning, involved in the production and evolution of a software system. A software process model is a perscriptive representation of software development activities in terms of their order of execution and resource management. The basic function of a process model of developing a software system is to describe the chain of events required to create and maintain a particular software product [11,8]. Software process models may also provide a basis for structuring software environments.[2]

Information relevant to design should be considered from three different aspects of a software system.

- From the point of a system structure, components of the desirable system and relation between them are considered. The concepts of procedures, modules and subsystems etc. are proposed and widely used.

- From the point of development process of a software system, researchers have proposed several well-known process model such as Waterfall model and Spiral Model.

- A very important, but sometimes neglected, aspect is the evolution of a software system. Version control and configuration management are among the existing concepts for controlling software change. However, these concepts only record the effects of change, they do not help manage change.

Although a number of models related to software development activities have been proposed which cope with the different information, how to organize the information from all these different aspects is still a problem which has not been fully considered. Therefore, we put forward a proposal for an organization schema which deals with the information pertaining to three aspects of design mentioned above. All the information from these three aspects will be organized in the schema around the decisions which are made.

Since the products being developed in a project, the processes which create and transform these products, and the organization and environment in which these processes occur have an interdependent progress, one fundamental problem is how to organize them in a schema which can be used conveniently to support later redevelopment activity.

In the following sections, we first describe previous work and give an outline of current research. Then the significance of this research topic is summarized and some of our work is given. Finally current directions of our research project are described briefly.

2 Previous Work

As we noted earlier, modeling the process of software engineering and evolution represents a promising approach toward understanding and supporting the development of large-scale software systems. In the following subsection, we will first discuss the previous works in the design process for initial development Then evolution models from the view of a software history are briefly described. Because design is a ill-structured, knowledge intensive problem solving activity, an AI approach to design is discussed in section 2.3

2.1 Design Process Model

A number of software process models have been proposed such as Waterfall model[3], Iterative Enhancement[12], the automation-based paradigm [10]. Some of the limitations of current models are the lack of support for project management, lack of visibility of design decisions and rationales and limited support for software evolution.

2.2 Software Evolution Models

A lrage software system usually experiences many changes over the course of tis lifetime. Large systems change gradually, in relatively small steps as it adpats to changing user's requirements and problems are fixed and . The direct effort of each step in the evolution of a software system is a change in one or more of the component objects comprising the system. These changes affect the functionality and the performance of the system as well as its representation and must respect many dependencies between the components to avoid damaging the system. Two of the main objectives of version management are:

- ensuring that consistency constraints are met.
- coordinating concurrent updates to subcomponents of a system.

Evolution steps can be represented as dependency relations between versions. There are often very many evolution steps in the life-time of a software system, and some of these steps may fork off new branches to create families of alternative versions of the system, which may differ in functionality or performance, and may interface to different operating system. peripherals. These alternate instantiations of the system and their supporting decisions offer a model of reuse based on componenet families distinguished by a set of controoling decisions. The complexity of this structure and the dependence of future changes on past design decisions make it important to record the evolution history of a system.

Current version control systems, such as SCCS and RCS, provide good support for keeping track of versions of files in a software system. But they provide only marginal support for understanding the structure of a large system consisting of many modules or subsystems and for keeping track of relations between documents, source codes, and test cases.

Unlike version control systems which only aim at identifying and efficiently storing many versions of software objects up to date, configuration management

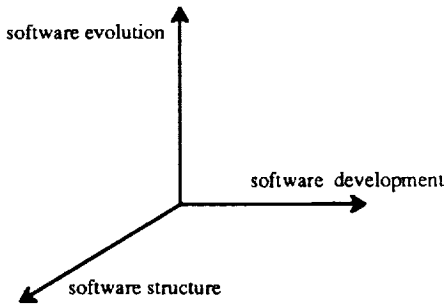


Figure 3.1 Three Dimensions in Software Development Process Space

Figure 1: Three Dimensions of Software Development

Goal structure of the design. Design is a purposive activity, goals guide the choice of what to do at each point. These goals are not artifact descriptions, but prescribe how those should be manipulated.

Design decisions. Given a goal, there may be several plans for achieving it. Design decisions represent choices among them.

Rationales for design decisions. The rationale for choosing a particular plan to achieve a goal explains why the plan is expected to work and why it was selected instead of other alternatives.

Control of the design process. Guiding design requires choosing which goal to work on at each point and choosing which plan to achieve it with.

Our research focus on the following aspects:

1. Capturing the design rationales of software design process in terms of decisions made during design process;
2. Developing a meta-model which may include major properties of contemporary software development practice, that is, consider development activities in a three dimension space; see figure 1
3. Advancing evaluation criteria for the development and evolution model.

Following the development process of a software system, traceability across the software life-cycle between

Figure 2: The Basic Model of 3DPM

view exactly those parts of a document impacted by a single decision. Conversely, when viewing a document, all the relevant decisions which impact that part of a document can be accessed. A process model for development and evolution within the DBSD paradigm is described in [14]

This model addresses the following:

Requirements Analysis. The purpose of this phase could be to identify the characteristics and requirements of the design problem.

Design Synthesis. involves exploring alternative solutions to the design problem.

Design Evaluation. developes the criteria for justifying a choice among a set of alternates.

Design Detail. Support implementing the chosen solution.

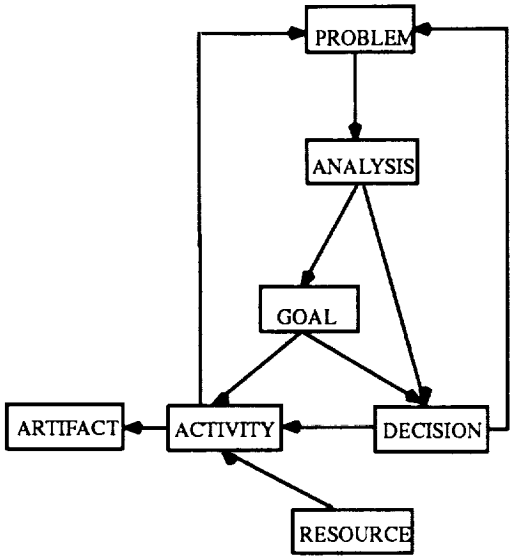
4.1 An Abstractive Structure of Model

Our experimental model is called $3DPM_p$ (three Dimensions Process Model - Prototype). A basic model is shown in figure 2.

A Problem is a description of the requirement or goal to be achieved.

Analysis of the problem given to identify all the possible alternatives and related justifications.

A Decision includes decision-making process and a final choice.


PROBLEM
ANALYSIS
GOAL
ARTIFACT
ACTIVITY
DECISION
RESOURCE

Resource is for organization management including resource allocation for any action.

Activity is a step for carrying out any outcome from decisions. It can not be activated until all necessary resources are available.

An Artifact is the product produced by the process.

4.2 Directions of Our Current Research

As described in [15], the DBSD paradigm is supported by a prototype CASE tool called D-HyperCase. Our current research focuses on the use of DBSD to facilitate reuse and to manage the development and evolution process. First of all, we should articulate what will be reused. In decision-based model, we will reuse not only products, but also decisions, justifications and alternates. The unit of reuse is a component family, where the term component denotes any collection of software function which can be considered as a solution to a generally recognized problem or set of related problems. The set of specific component instances comprises the family. Associated with a familty is a set of decisions, called the *component decision set*. Different members of the family are distinguished by the choices made for the set of decisions in that set. This is a different view of reuse then the traditional components of libraries approach. Generally, we have two different models for reuse. One is composition model (related to the abstraction reuse model of Campbell [7]) and the modification model.

In the compositional model, the set of alternates for each decision in the component decision set is kept in a library. This model involves the instantiation of each decision with a particular alternate and the composition of the chosen alternates into a fully instantiated family member. The justifications between decision serves to maintain consistency of the instantiated components and to explain the relationships among the set of decisions. This composition model will be used in our version control subsystem.

The composition model assumes data base populated with a rich set of alternate views and a set of rules of composition (in the form of a programming language for views or as an automated synthesis algorithm). In many cases, reusable components are organically grown with experience with its application. Although DBSD was orginally developed to support software maintenance, the same information structures support the modification model of reuse. In this model, an existing component is modified by changing one or more decisions. The decision view pinpoints those portions of the component which should be modified to support the new alternate solutions. This produces a new structure which instantiates another member of the component family. Some of the open problems under investigation are how to identify potentially reusable components, how to locate relevant decisions which need to changed, how to assess impact of changing those decision (that is, to assess the cost of reuse) and how to maintain consistency of the modified components.

We have developed a process model for management for decision-based software development paradigm. In this model, the decision review meeting is a key component. This meeting follows a structure which defines roles for each of the team members, the set of activities to be undertaken, the documents produced and the action items to be followed through after the meeting is over. The process model consists of several activities comprising the problem solving cycle.

A) Decision Review: The first step is the review of all actions taken since the last meeting. During this meeting the state of a problem can be changed in the following ways: problem is new, alternate solutions identified, rationale for choosing among the alternates given, decision made to solve this problem.

B) Knowledge Base Update: After the decision review meeting, the status of the problem is recorded in the *DBS* data base. Unresolved problems are placed on a problem agenda.

C) Resource Allocation: Management allocates resources to the problems on these agendas moving them into an active task list.

D) Implementation Chosen solutions to problems on the active task list are implemented.

The primary reason why the decision is the focus of our method is because of the key role decision play during software maintenance. More detail on the process model is given in [14].

5 Conclusions and Future Work

A software development process involves many kinds of information which should be considered in a three dimensions space called SDPS (Software Development Process Space) and organized around the concept decision.

Based on the previous work and experience learned from it, our work is planned as follows:

1. Analyze the related information entities in the design process, especially the design rationales, and put them into highly abstractive entity classes;
2. Study the possible relations between objects in three dimensions as well as the relations between the objects and decisions;
3. Use the technology of object-oriented database, hypermedia, and knowledge base to build 3DPM;
4. Study the relationship between 3DPM metamodel and other design approaches and explore the way of using this model in practice;
5. Develop metrics to evaluate the properties of this model and set some criteria to use it in various environments.

To support evaluation, we have developed the Decision, Instrument, ReEvaluate paradigm (DIRE) in which decisions which are weakly justified are identified as conditional decisions (that is, a decision is more likely ot change than other decisions). The artifacts which depend on these conditional decisions are instrumented to collect the data necessary to evaluate the quality of the decision and may lead to subsequent maintenance tasks to rework these conditional decisions. Metrics are used to support or refute these conditional decisions but may also indicate the situations under which different alternates are most appropriate within a component family. We are currently instrumenting a prototype CASE tool supporting the DBSD paradigm in order to development and validate 3PDM.

6 Acknowledgments

This work is supported by NASA under grant NAG1-966.

References

[1] A.Lie and etc. Change oriented versioning in a software engineering database. *ACM software Engineering Notes*, 14, no.7, Nov, 1989.

[2] R. Balzer. Process programming: passing into a new phase. *Proceedings of the 4th international software process workshop (May, 1988)*, 1989.

[3] B.W.Boehm. Software engineering. *IEEE Trans. on Computers*, 25,no.12, 1976.

[4] S. Dasgupta. The structure of design processes. *Advances in Computers*, 28, 1989.

[5] E.Horowitz and R. Williamson. Soddos: s software documentation support environment - its definition. *IEEE Trans. Software Engineering*, 12, no.8, August, 1986.

[6] V. Goel and P. Pirolli. Motivating the notion of generic design. *AI Magazine*, 10(1):118–38, Spring 1989.

[7] J. Grady Campbell. Abstraction-based reuse repositories. *Proc. AIAA Computers in Aerospace*, VII:368-373, October 1989.

[8] M. Jackson. Software development in the year 2000. *Proceedings of the 11th international conference on software engineering*, May, 1989.

[9] J. Mostow. Toward better models of the design process. *AI magazine*, 6 (1), Spring 1985.

[10] R.Balzer and etc. Software technology in the 1990's: using a nem paradigm. *Computer*, 16, no.11, 1983.

[11] W. Scacchi. Modelling software evolution: a knowledge-based approach. *Proceedings of 4th international software process workshop (May 1988)*, 1989.

[12] V.R.Basili and A.J.Turner. Iterative enhancement: a practical technique for software development. *IEEE Trans. Software Engineering*, 1,no.4, 1975.

[13] C. Wild and K. Maly. Decision-based software development: design and maintenance. *Proceedings of the Conference on Software Maintenance*, 297–306, October 1989.

[14] C. Wild, K. Maly, J. Dong, and G. Hu. A process model for decision based software development. *Proceedings of Conference on Software Maintenance*, submitted, Oct. 1991.

[15] C. Wild, K. Maly, and L. Liu. Decision-based software development. *Journal of Software Maintenance*, 3(1):–, 1991.

[16] C. Wild, K. Maly, and L. Liu. Decision-based-support-paradigm: a new method to structure source code. *Proceedings of the Conference on Software Maintenance*. –, November 1990.