

DETC2001/CIE-21233

USING THE ABSTRACT PROTOTYPING TECHNIQUE IN THE DEVELOPMENT OF DESIGN SUPPORT SYSTEMS

E. Z. Opiyo, I. Horváth, and J. S. M. Vergeest

Delft University of Technology, Department of Design Engineering,
Computer Aided Design and Engineering Section,
Jaffalaan 9, NL-2628 BX Delft, The Netherlands.
Email: e.opiyo@io.tudelft.nl

Keywords: Testing of design support tools, pre-implementation testing, prototyping, abstract prototyping.

ABSTRACT

In the process of development of Computer Aided Design (CAD) software tools, the expected functionality reappears in various different forms, for instance, as a collection of theories, methods, algorithms or as pilot implementations. Ensuring quality of these abstract software implementations can help assuring the acceptance of the functionality. The problem is that a formal methodology for assuring quality of all these abstract implementations is not available. As an attempt to deal with this challenge, we have developed a computer based pre-implementation testing methodology that has been named abstract prototyping. The exclusive feature of the abstract prototyping technique and the philosophy behind it is that it provides a framework for pre-implementation testing and for involving the representatives of various stakeholders in the development and testing of incidental implementations. We are using abstract prototyping as a framework for development and testing of abstract implementations of design support software tools, and it appears to be useful. It provides the developers with a methodology for exploring views of various stakeholders. It thus helps reduce the risk of developing poor design support software tools. In this paper we first review the problem and then briefly introduce the abstract prototyping concept and software tools. Afterwards, we focus on explaining how to use the abstract prototyping technique in the development and testing of design support software tools.

1. INTRODUCTION

Ensuring quality of the Computer Aided Design (CAD) systems is primarily the responsibility of the developers. In CAD systems development projects, the traditional software engineering approaches for requirements definition, implementation and testing are typically followed to assure quality. There are several classical software development

models (e.g. waterfall, rapid prototyping, program growth, etc.) and testing methods (e.g. reliability tests, performance tests, usability tests, etc.) in place. There are also several review techniques and standards. However, with all these in use, there are still many problems and dissatisfactions among the end-users. Various literature (e.g. Dekkers, 2000) report that many software development efforts do not succeed in delivering good quality products in time and within budgets. Often it is not until late stages of the development process, or even as late as after delivery, that the end-users can conclude that the software product does not meet their expectations. Moreover, the present testing techniques typically involve usage of partially or fully implemented prototypes. As a result, testing processes become lengthy and expensive, especially if the right solution is not achieved in the first trial. Performing evaluation prior to implementation of prototypes can help assuring quality, shorten development time and may reduce development costs. Techniques for software quality and process management that embrace aspects of pre-implementation testing such as CMM (Paulk et al, 1993) and usability tests (e.g. Whiteside, 1988) are available, but nevertheless most of them do not cover all phases of development. Our research work has resulted into creation of a comprehensive pre-implementation testing technique that has been named abstract prototyping.

In this paper, we first briefly explore the state of the art of design support tools development and testing practices, and concisely lay down the fundamental concepts of abstract prototyping. Then, we describe the abstract prototyping methodology and software tools that have been built to support developers in pre-implementation testing. We finally explain and illustrate, using a practical example, how the abstract prototyping methodology and software tools can be used in the processes of development and testing of abstract implementations of design support software tools.

2. REVIEW OF THE STATE OF THE ART

There are several approaches that are followed to assure quality of software products, including design support software tools. These include e.g. (i) assurance of the process by which the software product is developed; and (ii) testing of the prototype software implementation. However, there is little consensus among the developers of design support tools (and even the broad software engineering community) concerning the exact nature of the development or testing methods that are most appropriate (Moore, 2000). Most of the widely used development models e.g. waterfall, program growth and rapid prototyping include requirements analysis and specification, design, implementation, testing and installation, and operation and maintenance (Jones, 1990) as the main phases of the software development process. Consensus making and selection is an activity that recur in most phases of the software development process, e.g. at the design phase in which several alternative designs may be available and the developers have to choose the suitable design. Several creative thinking and/or consensus making systems have been proposed. Some of them are somewhat specific (e.g. those proposed in Kato and Kunifuji, 1997; and Kawakita, 1975, which are for consensus making during requirement acquisition and analysis), while others (e.g. decision matrices) can be used across the whole development process. There is little consent concerning which consensus making systems are most suitable. However, it is generally agreed that the decision matrix approach can provide reliable results, provided that right criteria are used and the appropriate subjects are involved. In the framework of most of the existing software development models, tests are typically done at the 'implementation' phase (at which parts of the programs are evaluated as they are completed) and at the 'testing and installation' phase (at which all parts of the software are put together and evaluated) (Bashir and Goel, 2000; Jones, 1990). Apparently, some of the abstract software implementations such as theories, methods and algorithms are developed, but are typically not formally tested.

Several software testing techniques have been reported in various literature (e.g. in Beizer, 1999). On the other hand, a number of techniques for software quality and process review are available. The Capability Maturity Model (CMM) (Paulk et al, 1993) is one of such review techniques. There are also several kinds of software tools for automated testing. These include, for instance, Software Engineering Environments (SEE), Integrated Project Support Environments (IPSE); Computer Aided Software Engineering (CASE); Meta CASE and Computer Aided Method Engineering (CAME) tools [Gray et al (2000)]. The requirements tracing facilities of automated testing methods typically provide capability to cross-reference any type of information to the modules, functions, data types, variables and constants in the model. Cross-referencing can therefore define the functions that implement quality requirements and conversely, quality requirements satisfied by functions. The problem is that not all quality requirements relate directly to the functions. The capability of the automated testing

tools to trace whether or not the quality requirements that are not connected to the functions are fulfilled is therefore limited.

In conclusion, one of the main problems with most of the conventional automated and non-automated testing approaches is that not all abstract software implementations are tested. For example, theories, which form the foundation of functionality, are in most cases not formally tested. Most tools are designed and built based on the existing knowledge. Similarly, the underlying methods and algorithms are typically not formally tested. Another problem is that the prevailing automated testing methods tend to distance the expertise of the developers and exclude the experience of the end-users from the test floor. On the other hand, the usability tests overwhelmingly leads to testing of the end-users requirements.

As an attempt to deal with the mentioned problems, we have developed a development and testing methodology that has been named abstract prototyping. It serves as a framework for involving representatives of various stakeholders of design support tools in testing of abstract software implementations, starting from the highest level of abstraction of the underlying theories, followed by the methods, algorithms, and pilot implementations. In the following section we briefly define some key concepts in abstract prototyping.

3. THE ABSTRACT PROTOTYPING CONCEPT

Abstract prototyping can be defined as the process of modeling and testing of abstract design support software implementations. We recognize theories, methods, algorithms and pilot implementations as testable implementations of the design support software tools. In the framework of the abstract prototyping technique, tests are done systematically, by involving representatives of various stakeholders (broadly categorized as the developers and the users) as subjects, as shown in Figure 1.

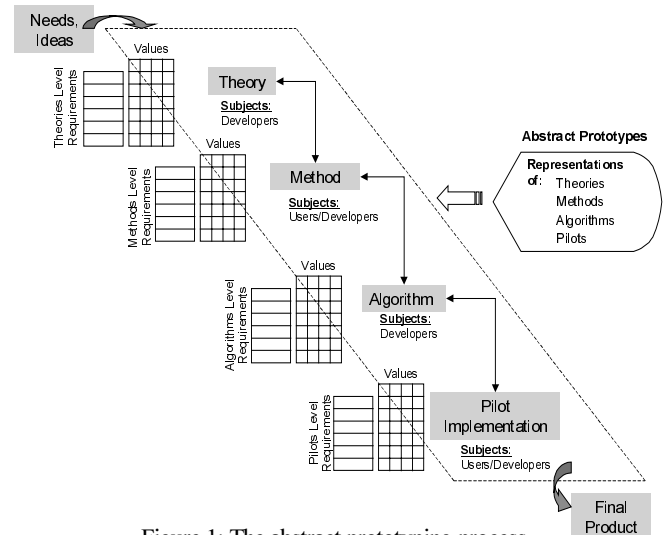


Figure 1: The abstract prototyping process

Abstract prototyping is essentially the symbiosis of a methodology and software tools usage in the development and testing of abstract software implementations. It furnishes the

developers with a methodology for exploration and reasoning about the alternative solutions during conceptualization of functionality of software (Figure 2) and systematically brings into the test floor various stakeholders of design support tools (i.e. various user groups, developers and other experts). It helps developers think far beyond their own experience and expertise and reach across stakeholders and other experts, to find solution to problems using knowledge and experience extracted from them through their involvement as the evaluation subjects. Abstract prototyping is even more than just a methodology. It is a way of thinking that enables developers to project and reflect the contents of their solution. It can be considered as a post-processor of the evolutionary design methods [such as TRIZ based solution finding (Altshuller, 1984; Altov, 1994)].

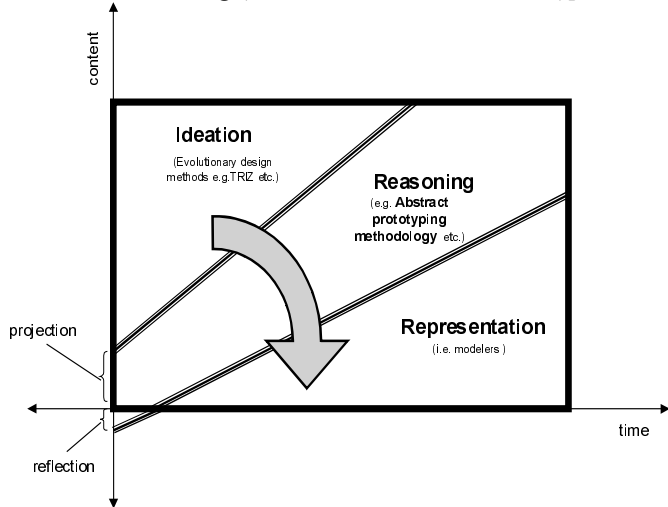


Figure 2: The role of abstract prototyping in the conceptualization process

The systematic schemes that constitutes the abstract prototyping methodology i.e. (i) the balanced comprehension scheme that guides ranking of alternative solutions, (ii) the stakeholders involvement scheme that guides the participation of the stakeholders in abstract prototyping, and (iii) the weak spots analysis methodology that helps finding weak spots in the solutions are described in detail in Opiyo (2000). Computer based methods for representation and processing of requirements and abstract prototypes, and for preparation of information gathering forms are also provided.

The principles of abstract prototyping can be summarized as follows:

- Theories, methods, algorithms and pilot implementations are testable prototypes at the life cycle stages of design support software tools. We refer to a system of these early implementations as *abstract software implementations* and the development stages as *levels of abstraction*.
- Requirements can be clustered according to these levels of abstraction, and

- The representatives of the stakeholders are systematically involved in the assessment of quality of the abstract software implementations as they evolve. Requirements are the evaluation criteria and are significantly improved through the elaboration processes that involve the evaluation of the representations of abstract software implementations, also called *abstract prototypes*.

4. THE ABSTRACT PROTOTYPING ACTIVITIES

Figure 3 shows the abstract prototyping procedure. There are five main activities, namely (i) requirements representation and processing; (ii) modeling, representation and processing of abstract prototypes; (iii) information gathering; (iv) selection of subjects, and (v) information analysis and interpretation.

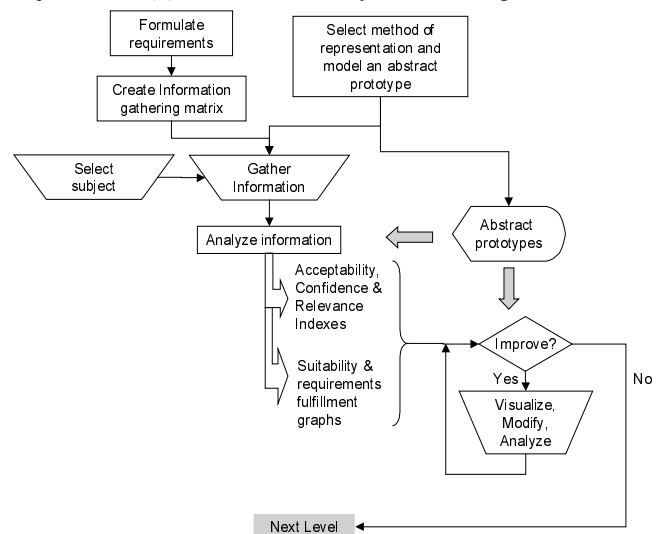


Figure 3: The abstract prototyping procedure

4.1 Requirements Representation and Processing

Requirements are modeled in advance (i.e. formulated from scratch in previous projects) and stored for possible future use. There are several ways of grouping requirements. These include, e.g. according to their sources, needs, levels of abstraction, functionality, whether the requirement is a functional or non-functional requirement, specific or general-purpose requirement, etc. There are huge links among the requirements, and even in some cases there are requirements that must be taken into consideration at all levels of abstraction. Figure 4 shows the relationships among the requirements (and hence needs) at various levels of abstraction.

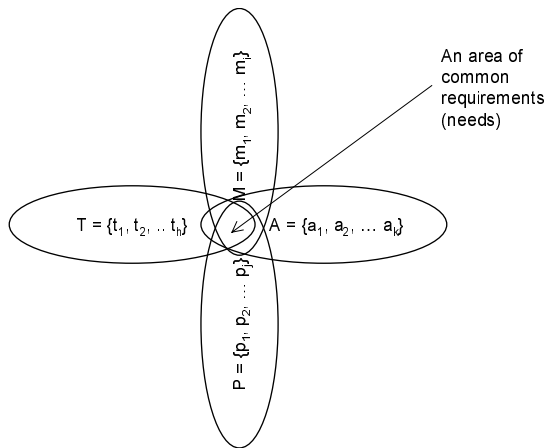


Figure 4: The relationships of requirements at theories, methods, algorithms and pilot implementation levels of abstraction for a functionality

T , M , A , and P are sets of requirements for the anticipated software at theories, methods, algorithms and pilot implementation levels of abstraction respectively. The intersection of T , M , A , and P is the area of the common requirements (and hence needs). This area represents the requirements that must be fulfilled at all levels of abstraction.

Figure 5 shows the methodology for formulation of requirements for a given task in the framework of the abstract prototyping technique.

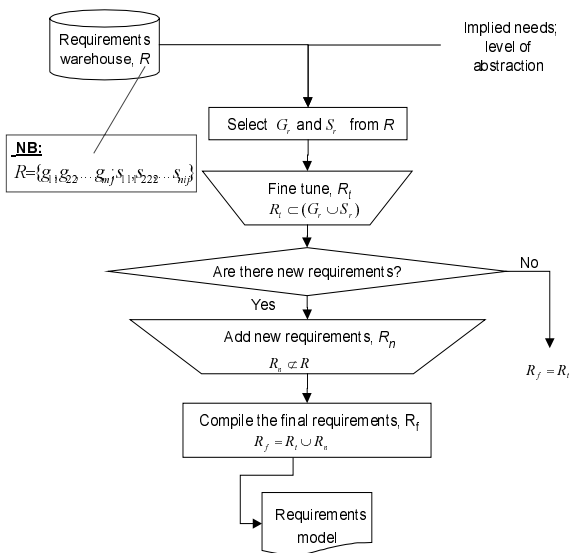


Figure 5: Formulation of requirements for a functionality

The starting point of the process of formulation of requirements for a functionality is the database, in which all kinds of 'old' requirements are stored. The requirement definition program takes as input a set of stated or implied

'needs' and the 'level of abstraction' and automatically produces a provisional requirements model, which is fine-tuned and ultimately used in the pre-implementation evaluation.

A function of a software can stand for one or more needs. Consequently, we are using 'functionality' as a keyword for searching requirements linked to particular needs of a functionality. For a given functionality f_i at a given level of abstraction l_j , sets of matching generic requirements $G_r = \{g_1, g_2, \dots, g_m\}$, $G_r \subset R$ and specific requirements $S_r = \{s_1, s_2, \dots, s_n\}$, $S_r \subset R$ are traced from the set of 'old' requirements R , where $R = \{g_{1p}, g_{22}, \dots, g_{mp}; s_{11}, s_{22}, \dots, s_{nj}\}$. The G_r and S_r are then fine-tuned (i.e. the requirements considered to be irrelevant are discarded) to match the functionality in question. A requirement model consisting of a set of 'fine-tuned' requirements R_f ; $R_f \subset (G_r \cup S_r)$ is generated. If the requirements are not as complete as they should be, then new additional requirements R_n ; $R_n \subset R$ may be formulated. The final list of requirements (R_f) to include in the information-gathering matrix (Opiyo, 2000) will then consist of 'fine-tuned' and 'new' requirements, i.e. $R_f = R_f \cup R_n$. Otherwise the information-gathering matrix will consist of the 'fine-tuned' requirements only, derived from the system's database.

The advantage of this approach is that the requirement formulation process is partly computerized and therefore allows the developers to participate or intervene. On the other hand, computerization of this process makes it less dependent on the composition or experience of the development team. The danger of overlooking requirements is also reduced.

4.2 Modeling, Representation, and Processing of Abstract Prototypes

The subjects, including novice subjects from outside the development team, should understand the abstract software implementations fairly clear for them to be able to express their views accurately. The abstract prototyping offers methods that help the developers search or model the representations of the abstract software implementations. It also provides tools that allow subjects to experiment and explore the implementations, and thus enhances the opportunity to make right judgements. Alternative ways of representation of chunks of knowledge about abstract prototypes are provided. At the theory level of abstraction, an abstract software implementation can be represented in text form, as an audio description, as a simplified model, or as a mathematical expression, while at the methods level, diagrams, text descriptions or procedural descriptions can be used. At the algorithms level, diagrams (i.e. structure charts and data flow diagrams), and pseudo codes can be used while at the pilot implementation level, an abstract software implementation can appear as a running (executable) program or as a pre-shot video clip. The purpose of availing the abstract

software implementations in different forms of representation is to help subjects to understand them better during evaluation. Representation templates are prepared in advance such that once an abstract prototype is thought of, it is modeled to match one of the templates.

The knowledge representation and processing utility also provides means for mapping an abstract prototype, say at a higher level of abstraction to those in the subsequent levels. This can be achieved as follows: Given, a theory t_j where; $t_j \in T$ and T is a set of theories for a given functionality, possible methods m_i where $m_i \in M$ and M is a set of available methods which satisfy specific requirements S_r can be found automatically. Similarly possible algorithms, a_i , where $a_i \in A$ and A is a set of available algorithms, and pilot implementations I_i where $i_i \in I$ and I is a set of available pilot implementations which satisfy specific requirements S , can also be determined automatically.

4.3 Selection of Subjects

The success of abstract prototyping depends on among other things the selection of subjects who assess quality of abstract software implementations. On the other hand, quality requirements of software products, including the design support software tools refers to given needs, which vary among stakeholders. Different stakeholders may have different functional and non-functional requirements. Consequently, right subjects means reliable results and vice versa. Typical software development projects have various stakeholders. In abstract prototyping, there are three broad classes of stakeholders, namely (i) the *users*, (ii) the *developers*, and (iii) the *managers*. The kinds of *users* to incorporate as subjects vary according to the task that software supports. For example, for the design support software tools, *users* may include draftsmen, design engineers, and other indirect users such as process planners. The expertise required in the development can be available within and from outside the development team. Consequently, the experts that should serve as subjects should be from within and/or outside the development team. Managers are typically interested in the overall product quality rather than in specific quality characteristics. Business related requirements are of more importance to them. They need to balance quality improvement with management criteria such as schedule delay and cost overrun (i.e. to optimize quality within limited cost, human resources and time-frame). A requirement model must address the needs of all stakeholders. A successful abstract prototyping exercise should ensure the incorporation of views of all stakeholders. Selection of subjects to include all stakeholders is an indirect way of counterchecking whether their requirements and therefore their needs are being taken into consideration.

4.4 Information Gathering

The tool used in the collection of views of the stakeholders in abstract prototyping is called *information-gathering matrix*. This is in essence a kind of decision matrix, which consists of an intuitive range of values and variables with which the fulfillment of a requirement can be judged. Subjects are required to input into the *information-gathering matrix* the values that represent their opinions. The value associated with a requirement statement constitutes to the measure of its level of fulfillment. For example, one requirement for a solid modeling functionality, which is very likely to be of interest, is the size of the library of primitive shapes it uses. Values that express a range of sizes can be assigned to it. Depending on the nature of the requirement statement, the following sorts of values may be used:

- *Binary values*: These apply to requirements that require a simple "Yes" or "No" value. For example, asking whether the solid modeling functionality allows the users to add their own personal library leads to a response whose value is simply 'Yes' or 'No'.
- *Classificatory values*: These values are for requirements that require generalized responses. For example, we might have a requirement, which states how a new standard shape can be added to a library. The range of possible values would be: inside the library, outside the library, and not possible.
- *Comparative values*: This is for requirements that require relative values. For example, 'similarity' can be important factor in user acceptance of a new piece of software, say, when developing a functionality that resemble one that has been successful in a different profession (e.g. design vs. medical). If the subject can guess how a particular function works, then s/he will be happier with the software. It is hard to imagine a measure for similarity other than a simple relative measure which awards a score on a rating scale.
- *Numerical/Metric values*: A response from the subject can also be a metric value. For example, when asking subjects about specific numerical values e.g. how many personal libraries can be defined.

Most probably, before the subject can assign a value to a requirement statement, s/he need to know *facts*, which can be discovered, say, e.g. by reading a technical literature or by watching a video clip on the computer screen. In other cases, the value depends on a *human judgement*. In yet others, some sorts of *test* need to be carried out before the subject can decide on a value to assign to a requirement statement. This requires an experiment to be set and subjects to have some sort of hand-on experience prior to giving their opinion. The values assigned to requirements by the subjects ultimately give rise to quantifiable measures of the acceptability and the levels of fulfillment of requirements by the abstract software implementations.

4.5 Information Analysis and Interpretation

The values in the information gathering matrices are compiled and analyzed, and the results presented in tabular or graphical forms. Two measures, namely; (i) the acceptability, and (ii) the requirement fulfillment indexes must be determined. What needs to be achieved is to maximize the acceptability index and the levels of fulfillment of requirements i.e.:

(a) Maximize the acceptability function:

$$AI = \frac{\sum_{j=1}^n w_j \sum_{i=1}^m M_{ij}}{E_s};$$

subject to: $AI \geq AI_{thr}$; and

(b) Maximize the requirement fulfillment function:

$$F_j = \frac{\sum_{i=1}^n M_{ij}}{\sum_{i=1}^n F_m} \times 100\%$$

subject to: $F_j \geq F_{thr}$

where:

AI = the acceptability index for a solution proposal.

AI_{thr} = threshold acceptability index value.

E_s = maximum possible score.

F_j = fulfillment index for a requirement j .

F_m = the maximum possible fulfillment value.

F_{thr} = threshold fulfillment index value.

M_{ij} = score for a requirement j assigned by subject i .

W_j = a consensus weights (Opiyo et al, 2001), jointly assigned by subject i and the developers.

The acceptability index provides means for ranking the solution proposals, while the requirement fulfillment indexes are used in plotting the requirements fulfillment diagram. This diagram provides graphical means for recognizing the requirements that have not been fulfilled adequately. Further details on information analysis in the framework of the abstract prototyping technique are available in (Opiyo et al, 2000).

5. THE ABSTRACT PROTOTYPING SYSTEM

The abstract prototyping software portfolio includes utilities for representation and processing of requirements, representation and processing of abstract prototypes, preparation of information gathering tools, information analysis, solutions finding, and system management (Figure 6). The abstract prototyping software has been implemented based on the principles of abstract prototyping covered in e.g. Opiyo et al, 2000.

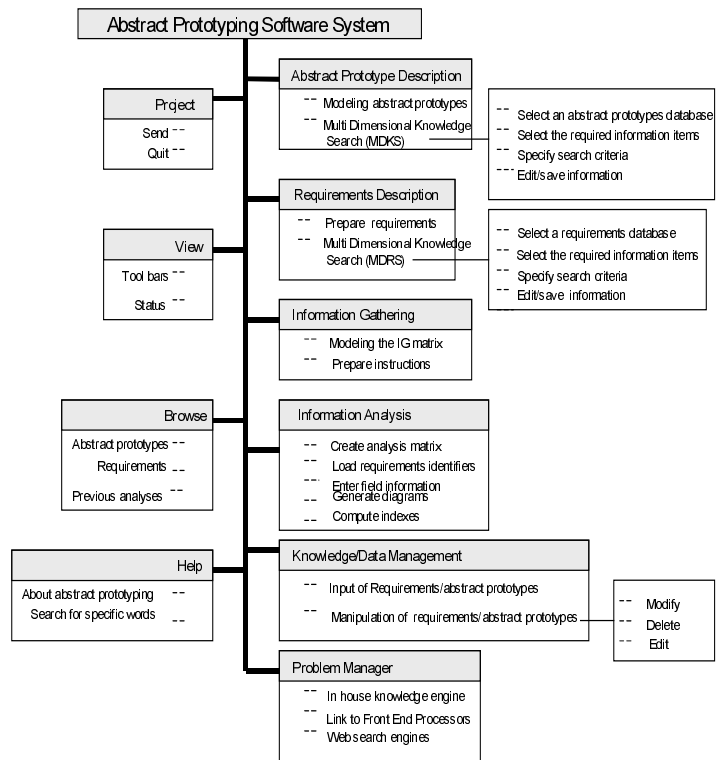


Figure 6: Utilities of the abstract prototyping system

The utility for representation and processing of requirements includes a multi-dimensional requirements search (MDRS) tools, which offers means for searching requirements based on criteria put forward by the developers (Opiyo et al, 2001). The abstract prototypes, representation and processing module includes among others a multi-dimensional knowledge search (MDKS) tool, which furnishes the developers with a functionality for searching knowledge about the abstract software implementations based on one or more constraints specified by the developers (Opiyo et al, 2001). A database (Figure 7) is provided to house information about requirements and abstract software implementations.

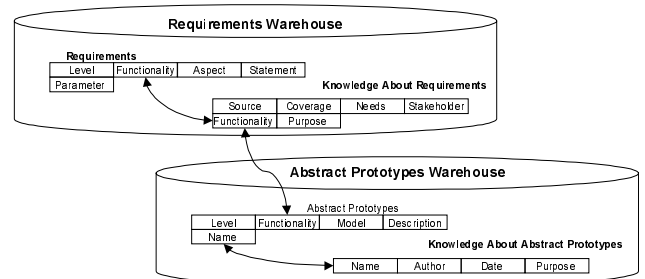


Figure 7: The APS warehouses

Abstract prototyping provides a technology that enables the development teams to (i) reuse requirements, (ii) model abstract software implementations, (iii) work in geographically distributed locations, and (iii) perform analysis of views of subjects (i.e. managing complex data and ideas, searching and exploring context, seeking patterns and meanings, and representation of results). Furthermore, it (iv) offers a methodology for testing abstract software implementations and relating them to requirements, and (v) leverages the intellectual assets of the developers by availing knowledge about abstract software implementations, thus allowing them to explore and develop design support tools of higher quality.

6. HOW TO USE THE ABSTRACT PROTOTYPING TECHNIQUE

In this section, we explain how the developers of design support systems can use the abstract prototyping technique. To test abstract implementations in the framework of the abstract prototyping technique, at a given level of abstraction, using the abstract prototyping software system (Figure 8), the developers have to stick to the following steps, also depicted in Figure 3:

- Having known the '*functionality*' and the '*level of abstraction*', use the WAREHOUSES facilities to find the requirements in the database.
- Fine-tune the requirements list (i.e. omit the irrelevant requirements and add missing requirements) and save the resulting requirements model in the WAREHOUSE.
- Use the REQUIREMENTS DESCRIPTION utility to convert the requirement model into a usable format.
- Use facilities in the ABSTRACT PROTOTYPES DESCRIPTION utility to (i) search for the relevant abstract prototype model in the database, (ii) edit it, and (iii) save the edited model.
- Use the INFORMATION GATHERING utility to send to the subjects the 'Information Gathering Matrix' and the 'Abstract Prototype' model.
- Use the INFORMATION ANALYSIS utility tools to:
 - [a] Create the analysis matrix,
 - [b] Input the requirements identifiers,
 - [c] Input responses from subjects,
 - [d] Determine the acceptability index,
 - [e] Generate the requirements fulfillment diagram, and
 - [f] Perform '*connections analysis*' (i.e. investigate the relationships among the implemented parts).

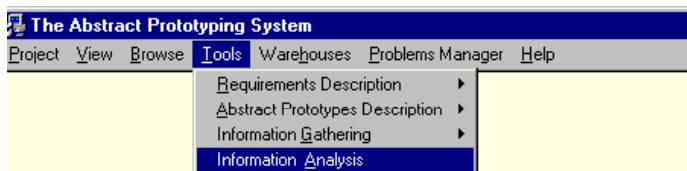


Figure 8: The abstract prototyping system window (before log-in to specific utilities)

- Use the WAREHOUSES utility to investigate the implications of choosing alternative solutions i.e. e.g. if, say, a certain theory is chosen, what are the possible methods, algorithms, and pilot implementations? This helps foreseeing beforehand whether, say, a chosen theory will, say, be implementable, etc.
- An in-house knowledge engine, web search engines, and front-end processors in the PROBLEMS MANAGER utility provide means for finding possible alternative solutions and for exploring their appropriateness.

It is important to mention that some preparations must be made beforehand. Such preparations include development of the rating scales and selection of subjects. The end product of abstract prototyping is a summary of data about (a) the levels of fulfillment of quality characteristics, and (b) ranking of alternative solution proposals. The requirements fulfillment diagram is also generated. This diagram shows the extents of fulfillment of requirements. Furthermore, an index called the '*acceptability index*' is also computed. It gives clues on the levels of acceptability the abstract software implementations. The qualifying alternatives are then visualized, modified and analyzed recursively until it is assured that all key requirements have been satisfied to a satisfactory level. The final decision on whether to keep on improving or to move to the next level of implementation is taken collectively by the developers based on their expert judgments and expertise.

7. VALIDITY AND RELIABILITY OF ABSTRACT PROTOTYPING

The validity of the abstract prototyping measures is a critical issue. If the measures are not valid, the evaluation as a whole is worthless. Both the measures and the methods used to obtain a measurement must be valid and reliable; that is, a range of values and variables should measure what it is supposed to measure, and should do so consistently. If there is controversy about the evaluations, it is often the validity of the measures or of the methods used to obtain a measurement which is called into question. There are several conceptions of validity in literature. They fall under one of two broad categories, namely, *internal* validity and *external* (or criteria based) validity.

The internal validity is achieved by making sure that what has been measured reflected an appropriate aspect of the object that is being evaluated. In the context of abstract prototyping, internal validity reflects the degree to which the measure represents views of the subjects. Ensuring internal validity is difficult to achieve. It relies on the judgement of experts, in our case, the judgement of those who design the evaluation (i.e. the developers), and can only be justified after the event, in the light of feedback from the end-users or from other interested outsiders. External validity is achieved by demonstrating a correlation between a measure and an external criterion. It is rarely worked out formally (by e.g., calculating a coefficient of correlation), but is often used informally to justify the choice of

a measure. It depends on the knowledge and experiences that those who design the evaluation (i.e. the developers) possess.

The measures must also be reliable. We call a measure reliable if a consistent method for achieving the measurement exist and if the same results are obtained each time the same measure is applied to the same object of evaluation. Reliability can be determined by calculating the coefficient of correlation between the results obtained on two or more occurrences of an experiment. Reliability is very difficult to check, especially when the developers intervene. The developers tend to have subjective opinions on their implementations. In the interests of reliability, it is advisable to limit the developer's intervention as much as possible. However, it is often inevitable that some measures will rely on developers' involvement e.g. in assigning weights to requirements. It is important to choose an appropriate method for carrying out the measurements. Such a method should offer some general proof of appropriateness, based on previous experiences. As far as abstract prototyping is concerned, it includes software methods for computation of the measures. This assures the reliability of the measures.

8. A CASE APPLICATION

Our group is involved with a research on development of a shape conceptualization system. Four modules of this system, namely, the highly interactive communication, natural internal representation, physical concept modeling and design knowledge handling modules (Figure 9), are under development. We aim at providing the designers with: (i) means for communicating shape information to computers in a natural way; (ii) modeling techniques that can cope with inconsistencies or contradicting input information; (iii) techniques for preparation of manufacturing information for production of large-sized physical concept models using the available manufacturing technologies; and (iv) techniques for handling of knowledge related to various design aspects such as aesthetics, cost and manufacturability based on ontology paradigm.

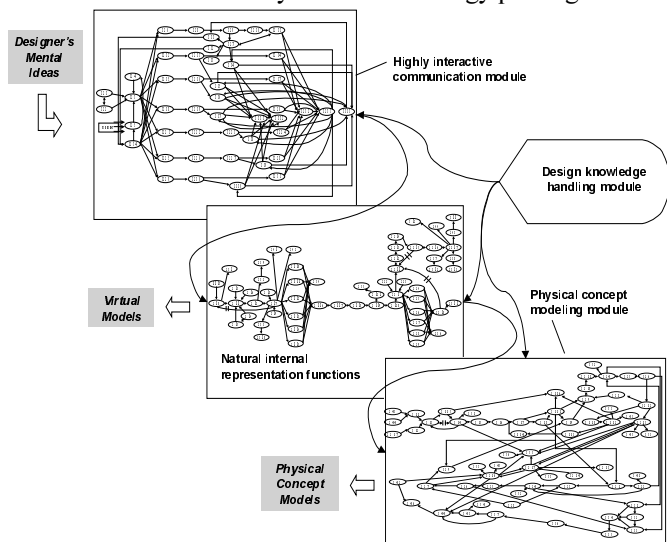


Figure 9: Functional representation of the shape design

There are many theories and methods involved, and in some cases several alternatives can be available for a single functionality. There can also be several ways for developing an algorithm for a single functionality. Our ultimate goal is to come up with effective and usable techniques.

We are using abstract prototyping as a technique for testing and selecting alternative solutions (i.e. abstract implementations). Due to space limitations, we present only one practical example to illustrate how we are using this technique. One team in our research group is involved with the development of highly interactive input means for shape design. They used abstract prototyping technique to investigate the preferences of designers on possible input techniques (table 1). The primary question was: Which combinations of input means are most effective in the opinion of the selected subjects?

Table 1: Possible input streams for shape design

Input streams	State of the art
Mouse and Keyboard	Widely used with conventional CAD systems
Graphic Pen and Keyboard	Not widely used, only occasionally
Voice Control, Gestures and Keyboard	Not used in shape design

Several aspects were taken into consideration in the evaluation of these shape-input methods. They took into account standards such as ISO 9126 and IEEE that are primarily concerned with the definition of quality characteristics used in the evaluation of software products. Standards set out quality characteristics such as functionality, reliability, usability, efficiency, maintainability and portability that describe software quality and other desirable characteristics as well. From the abstract prototyping system's database, the developers could also find several other non-standard desirable characteristics.

Using the multi-dimensional requirement search (MDRS) utility, the provisional list of 47 requirement statements connected to ISO 9126 quality characteristics and other characteristics were downloaded from the system database. The list was then fine-tuned to make it specific to this pre-implementation evaluation (i.e. irrelevant requirement statements discarded, in some cases the wording of the statements changed and one completely new requirement statement added). Table 2 shows the final list of requirements that were ultimately used in the pre-implementation evaluation.

Table 2: Requirements for the shape-input functionality (i.e. includes ISO 9126 and other quality characteristics)

IDENTIFIER	ASPECT	REQUIREMENT STATEMENT
R1	Privacy	The method allows for a designer or a team of designers to work without interference
R2	Usability	The method match natural ways of working
R3	Intuitivity	Users can understand how the input device works
R4	Efficiency	The input method is good for the intended task
R5	Training	The input method can be used without the need for

		extensive familiarization or learning
R6	Ergonomics	The ergonomics of the methods acceptable
R7	Portability	The input stream consist of portable devices
R8	Cost	The input devices are affordable
R9	Marketing	There is no marketing problem
R10	Reliability	The input method can provides consistent results
R11	Functionality	The method can solve the problem in question
R12	Efficiency	The method can be effective in actual use
R13	Cost	If adopted, the method will be operational.
R14	Cost	If is adopted, the method will be maintainable.
R15	Applicability	This input method can be used in the design of industrial products
R16	Benefit	The input method will comparatively brings along more useful features over the competing methods
R17	Shape information	The input stream can be used to communicate non-geometric shape information
R18	Shape information	The input stream can be used to communicate geometric shape information
R19	Versatility	The input stream can be used in inputting a wide range of shapes, including free-form shapes
R20	Shape information	With the input stream, shapes can be modified.

18 subjects, all of them industrial designers participated in the evaluation. The collected information was analyzed to (i) determine the levels of fulfillment of requirements, and (ii) establish the subjects' preferences. The acceptability indexes (Figure 10) for the alternative input streams were determined and the requirements fulfillment diagrams (Figure 11) plotted.

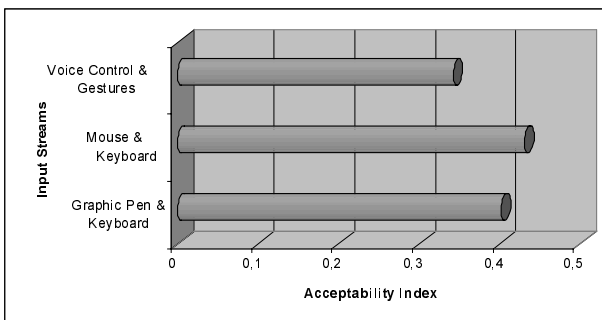


Figure 10: Acceptability Indexes for possible alternative input streams

In the opinion of the subjects that participated in this abstract prototyping exercise, mouse and keyboard are preferred most, but nevertheless with a slender margin. The (i) graphic pen and keyboard, and (ii) voice control, gestures and keyboard follow. The differences among the values of acceptability index for the input streams are almost insignificant. Even the trends in the levels of fulfillment of requirements do not significantly differ. It can be concluded that in the opinion of the selected panel, any of the three input streams are acceptable in one way or another. Voice control and gesture, and to some extent graphic pen method are not very common among the designers. Most designers cannot imagine how they can work with them, and how they will affect their working methods. This may explain why they are somehow less preferred.

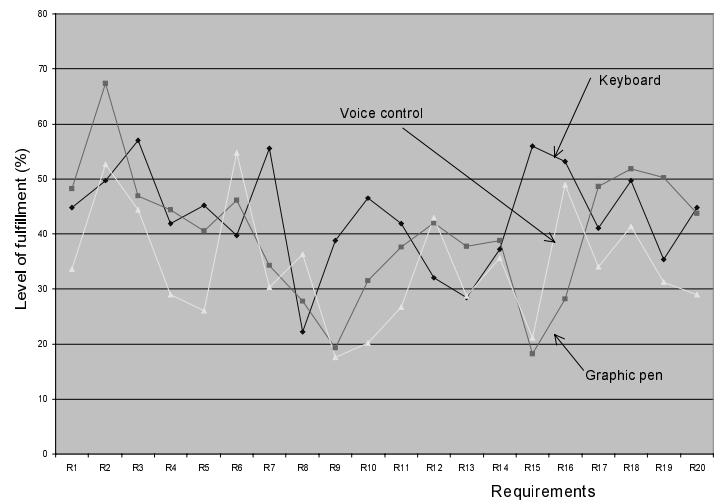


Figure 11: The levels of fulfillment of requirements

This experiment gave a clue on how the input stream consisting of voice control, gestures and keyboard is relatively preferred. In the opinion of the subjects, this input stream had an edge over the others in satisfying requirements such as R2, R6, R8, R12 and R16. The team could then start the next stage of development i.e. developing and testing algorithms with the developers as subjects, followed by implementation and testing of pilot prototypes by involving the designers as subject.

9. DISCUSSION

It is important to mention that the case application covered in the previous section illustrates only one way of using the abstract prototyping technique. There are three ways in which the abstract prototyping technique can be used in the processes of development of design support systems, namely, (i) as a stand alone development model; (ii) as a testing technique within the prevailing software development models; and/or (iii) as a technique for keeping requirements aboard during development. Pragmatically speaking, one may not be able to utilize or need all abstract prototyping methods at a go. Instead, depending on the needs and the size of the project, some of the methods can be utilized, ignored, or customized.

In using the abstract prototyping technique as a stand-alone development model, the development process should yield the following testable incidental abstract implementations: (i) foundational theories; (ii) methods (i.e. based on the selected theories); (iii) algorithms; and (iv) pilot implementations. Several solution proposals (i.e. abstract implementations) often come into the picture, especially at the higher levels of abstraction, i.e. at the theories and the methods levels. Pre-implementation testing of such alternative implementations and selection of the best alternative can be done in the framework of the abstract prototyping technique. Algorithms and pilot implementations are also testable implementations that often appear in the frameworks of the traditional software development models such as the waterfall and program growth models. These abstract software implementations can be tested

in the framework of the abstract prototyping technique. In using abstract prototyping as a technique for keeping requirements aboard during development, the requirements description utility provides software tools for multi criteria search of requirements and for cross-referencing requirements to functionality, level of abstraction and their origin.

There is suggestion in some literature, e.g. in Schluter, (1999) that points out that the use of standard development and testing methods is necessary, but slows down the development process. Apparently, time consumption of the use of methods is generally uncertain. It is important to mention that with regard to the abstract prototyping technique, there are open questions on the way of maximizing its efficiency. At the end of the day, the effects of the use of the abstract prototyping technique should outweigh those of not using it.

10. CONCLUSIONS

A computer based pre-implementation testing methodology, called abstract prototyping has been proposed and is being used in the process of development and testing of abstract implementations of design support software tools. Typical applications have shown that it effectively supports the developers in exploring the suitability of the implementations at various levels of abstraction. The unique feature of this methodology is that it provides a framework for involving the representatives of the stakeholders in assessing the consistence, completeness and usability of abstract implementations of design support software tools as they evolve. It gives them a chance to express their opinions during development, thus reducing the risk of developing substandard tools. Co-development and stakeholders oriented pre-implementation testing increase the chance of creating acceptable design support software tools.

ACKNOWLEDGMENTS

The research work reported in this paper relates to the Integrated Concept Advancement (ICA) project of the Faculty of Design, Engineering, and Production of the Delft University of Technology, Delft, The Netherlands.

REFERENCES

- Altov. H. "And Suddenly the Inventor Appeared", Worcester, MA: Technical Innovation Center, 1994.
- Altshuller. G. S. "Creativity as an Exact Science: The Theory of the Solution of Inventive Problems", New York: Gordon and Breach, 1984.
- Bashir, I. and Goel, A. L.: "Testing of Object-Oriented Software: Life Cycle", *Quality Techniques Newsletter*, Software Research Inc., San Francisco, CA USA. <http://www.soft.com/News/QTN-Online>, October 2000.
- Beizer, B. (1999). Testing: Best and Worst Practices. *Quality Techniques Newsletter*, Software Research Inc., San Francisco, CA USA. <http://www.soft.com/News/QTN-Online>, November, 1999.

Dekkers, M., (2000), "Making 'IT' Work", *Quality Techniques Newsletter*, Software Research Inc., San Francisco, CA USA. <http://www.soft.com/News/QTN-Online>, April 2000.

Giessen, M. and Sevenstern, B." Investigation of Designers' Input Devices Preferences", *IDE 350 Research Report*, Delft University of Technology, pp. 1-18, 2000.

Gray, J. P., Liu, A., and Scott, A., 2000, "Editorial - Special Issue on Constructing Software Engineering Tools", *Information and Software Technology Journal*, Elsevier, Vol. 42, pp. 71-72.

Jones, W. G. "Software Engineering", John Wiley and Sons, 1990.

Kato, N. and Kunifuji, S. (1997) "Consensus-making support system for reactive problem solving", *Journal of Knowledge-Based Systems* 10 (1997), pp. 59-66.

Kawakita, J., "The KJ Method: A Scientific Approach to Problem Solving", Kawakita Research Institute, 1975.

Moore, D. L., (2000), "Managing Requirements: From Battleship to Heat-Seeking Missile", *Quality Techniques Newsletter*, Software Research Inc., San Francisco, CA USA. <http://www.soft.com/News/QTN-Online>, November 2000.

Opiyo, E. Z., Horváth, and Vergeest, J. S. M., " Knowledge Representation and Processing in Abstract Prototyping of Design Support Tools", *International Conference on Engineering Design*, ICED 01, Glasgow, UK, In press - 2001.

Opiyo, E. Z., Horváth, and Vergeest, J. S. M., " Software Tools for Abstract Prototyping of Design Support Tools", *20th Computers and Information in Engineering (CIE) Conference: 2000 ASME DETC Conferences*, September 10-13, 2000, Baltimore, Maryland, USA, Paper No. DETC/CIE-14613; ISBN 0-7918-3506-5; New York, NY, USA.

Paulk, M. C., Curtis, B., Chrissis, M., B. and Weber, C. V., "Capability Maturity Model, Version 1.1," *IEEE Software*, Vol. 10, No. 4, July 1993, pp. 18-27.

Schluter, A. "Requirements on the use of Methods in Industry", *Proceedings of the International Conference on Engineering Design*, ICED 99, August 23-28, 1999, Munich, Germany, pp. 231-232.

Whiteside, J., Bennett, J., and Holtzblatt, K., 1988, "Usability Engineering: Our experience and evolution", *In Handbook of Human Computer Interaction* (M. Helander ed), Elsevier Science Publishers.