UNIVERSITY OF ZIELONA GÓRA
FACULTY OF MECHANICAL ENGINEERING
in association with the Design Society

THIRD INTERNATIONAL
SEMINAR AND WORKSHOP
10th - 12th October 2002
Zielona Góra - Łagów
POLAND

ENGINEERING DESIGN IN INTEGRATED PRODUCT DEVELOPMENT
Design Methods that Work

A STRATEGY FOR QUALITY ASSURANCE OF COMPUTER BASED DESIGN METHODS

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

Delft University of Technology
Department of Design Engineering
e-mail: {e.opiyo;i.horvath;j.s.m.vergeest}@io.tudelft.nl

Keywords: CAD/CAM/CAE, quality assurance, pre-implementation testing, abstract prototyping.

Abstract: *In software development processes, reviews are typically conducted to remove faults before requirements or designs are passed to the subsequent phase. However, in spite of the stringent review procedures, the process is still porous and significant numbers of faults conceals. This paper presents a novel strategy for reducing faults by assuring quality of the in-process implementations, dubbed abstract prototyping. It extends the current practices by defining the steps of the design phase of the processes of development of engineering design software tools. Under this procedure, reviews are performed to remove faults before theories, methods, algorithms, or pilot prototypes are passed to the subsequent stage rather than exclusively reviewing the requirements or designs. Prototypes provide the feel and the look of these in-process implementations and specially designed metrics help the developers estimate the extent to which they fulfill their respective requirements. Case studies show that the levels of fulfillment of requirements can adequately be estimated and faults detected early on.*

1. INRODUCTION

Quality is defined as the degree of excellence of something [1]. This implies that any software project can be found lacking if measured against an unclear notion of what quality is [2]. Various testing strategies can be used to determine whether software satisfy its specification, and it is recognized, for example, in [3] that each technique provides varying amount of assurance. It is universally understood that quality of software products, including those used in engineering design [variously known as Computer Aided Design (CAD), Computer Aided Design and Manufacturing (CAD/CAM) and Computer Aided Engineering (CAE) systems] is determined during the entire development interval [4]. Two strategies, namely (i) assurance of the process by which software is developed and (ii) verification and validation of various in-process implementations, are used to assure quality of software product. The former strategy include, for instance, using traditional software development models such as waterfall and rapid prototyping, abiding by standards, and following software process management techniques such as Rational Unified Process (RUP) and Capability Maturity Model (CMM). The later include conducting traditional tests such as unit tests, system tests, integration tests, and usability tests; carrying out reviews (that is, surveys, analytical studies, experimenting with prototypes, and so forth); and adhering to software quality standards such as ISO and IEEE standards. Most strategies are of post coding nature, while a handful of them are not. Despite of the presence of many strategies for assuring quality, still there are pitfalls and faults occasionally pass through the development processes and end up as bugs in the delivered software. In general, there is no consensus among the broad software engineering community concerning the exact nature of the development models, tests or even combinations that are appropriate for all tasks and software projects. Practitioners and researchers

concede that there is no silver bullet development model or verification and validation technique that will work for all software organizations and for all software projects [2], [5], [6], [7], and [8]. It is also generally understood that bugs cannot completely be eliminated [9]. In a typical software process, the design phase accounts for a significant net additional faults [10-11], and it is tempting to suggest that there is a need for a more robust strategy for assuring quality in the design interval.

In most cases the traditional software process models are used to direct and guide software development activities. They are typically multi phased, and consist of requirements, design, implementation, testing and operation as the main phases [12-13]. One of the goals of phased development is to minimize faults in the delivered codes. Many of the faults can be traced back to the requirements or designs. Reviews are typically performed to remove faults before the requirements or designs are passed to the subsequent phase. However, the review processes are porous and still faults passes into the subsequent phases. Part of the problem is that the activities in the phases of software process models are coarsely defined and do not scale to precisely match the needs of the actual processes. The design phases are typically broad, and in the course of creating final designs, different kinds of intermediate products that can be prototyped and reviewed, and consequently contribute towards reducing faults come into the picture. One of the possible measures to improve quality of software is therefore through prototyping and review of all design phase in-process implementations.

The CAD/CAM/CAE systems are different from other software in that they are based on engineering principles and methods. The design phase of these software tools consists of ad hoc activities such as development or specification of foundational theories, underlying methods, development of algorithms, and pilot implementation. Having known the needs and the constraints on the solutions, the developers of these software tools define features of the envisaged software and formulate or select appropriate theories for each feature. Based on the defined theories, the underlying methods are developed and reviewed. The algorithms are subsequently developed and codes for the key features of software written and tested. The initial activities of formulation of theories and methods are rather highly informal and happen intuitively. As an attempt to formalize this procedure and enable more reviews and tests to be done in the design interval, we have developed a pre-implementation testing methodology called *abstract prototyping* (abbreviated as *AP* in this article). It emulates and extends the current ad hoc practices, and defines the stages of the design phase of the processes of development of CAD/CAM/CAE software as creation or selection of theories, formulation of methods, design of algorithms, and writing codes for pilot prototypes. Based on this methodology, faults can be traced back to the requirements, theories, methods, algorithms, or pilot prototypes rather than exclusively to the requirements or the design phase end product (namely the design document in its final form). Under this procedure, prototypes are built and reviews conducted to remove faults before the requirements, theories, methods, algorithms, or pilot prototypes are passed to the subsequent level. Specially designed metrics provides means to estimate the extent to which these design phase in-process implementations fulfill the requirements.

In this paper, we present and discuss practical case studies on the application of the AP strategy. This is preceded by a concise overview of the background research.

2. BACKGROUND RESERACH

This section briefly describes of the theoretical fundamentals of AP, the methodology and software tools that have been developed to support software developers in reviewing the in-process implementations.

2.1. Abstract Prototyping

In the development of CAD/CAM/CAE software, it is important to ensure that right engineering principles (theories), methods, algorithms are deployed. It is also important to detect and eliminate flaws as early as possible, preferably prior to coding. This is because if a solution concept is changed after coding, then large sections of the code may have to be rewritten. The AP concept has been developed to ensure that appropriate theories, methods and algorithms are used. It is also meant to facilitate discovery of faults at the design phase, and it provides a way for proofing the in-process implementations at the design phase, before codes are written.

2.1.1. Fundamentals

AP can be defined as the process of prototyping and reviewing the in-process implementations at the design interval. Theories, methods, algorithms and pilot prototypes are regarded as testable implementations during the conceptualization and design of CAD/CAM/CAE software. AP is a staged process, and at each stage, the key objective is to minimize the number of flaws passed to the next phase. For the initial stages - namely theories, methods, and algorithms - at these moments the codes have not yet been written, and to ensure that the in-process implementations are of sufficient quality, reviews are conducted. At the pilot prototypes stage, codes are available, and apart from reviews, there are many other well-known testing techniques that can be used to find bugs. In the framework of the AP technique, reviews are done systematically, by involving representatives of various stakeholders, broadly categorized as the developers and the users as subjects. Theories and algorithms are highly technical in-process implementations and the developers are well suited to

serve as subjects during their reviews. On the other hand, the representatives of the users review methods and pilot prototypes, which are rather less technical and easily perceivable. Prototypes of the in-process implementations (called abstract prototypes) are built and provide the basis for discussing and clarifying the implementations. A review set-up typically consists of a panel of the representatives of the stakeholders' community, who give opinions on the implementations. The opinions are subsequently analyzed, and based on this, flaws in the in-process implementations can be identified. Then the developers rework the implementation and review it for the second time (if necessary) before the implementation is passed to the next stage (unless the re-identified faults are so severe and another rework and review is needed). The idea is to ensure that all faults are caught and eliminated as early as possible.
Fig. 1 illustrates various kinds of possible transformations between acquisition of requirements and realization of the expected software product in the AP context and how various stakeholders are involved in the reviews. The main path is the Theories-Methods-Algorithms-Pilot prototypes-Expected functionality (TMAPE) path. Having known the suitable theories, they can be transformed into methods, followed by algorithms, and finally into codes for pilot prototypes or the expected functionality. The TMAPE path guides the design process through the basic preliminary guises and the building blocks of the eventual software of different natures. Obviously there are other transformation scenarios, for instance, starting straight away to write codes for the expected functionality based on theories (TE), methods (TME), or algorithms (TMAE) and vice versa. There are also short cuts such as, writing codes for the expected functionality exclusively based on methods (ME), algorithms (AE), theories, algorithms and pilot prototypes (TAPE), theories and pilot prototypes (TPE), or methods and pilot prototypes (MPE).

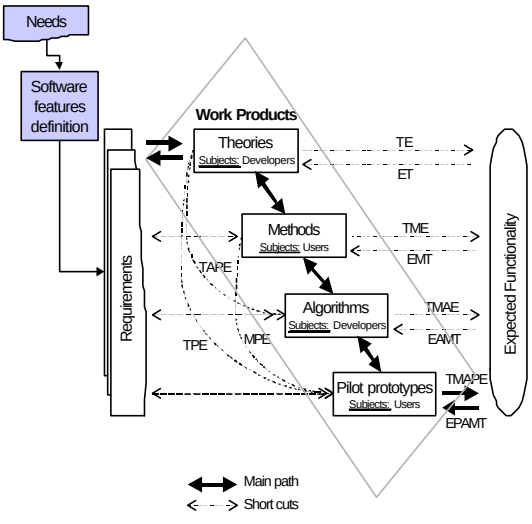

Fig.1. *A general scheme for realization of software in the framework of the AP technique*

In the development of novel functionality, it can, however, be detrimental, say, to jump directly into the development of methods, algorithms or pilot prototypes based on the understanding of requirements (or needs) only, without building a foundation on a theory or set of theories. Such practices can increase the rework cycles. The processes tend to be rather ad hoc, undisciplined, and of trial and error in nature, and thus the outcomes are prone to faults. On the other hand, the designs resulting from the TMAPE track are less vulnerable to errors since when gaps in the in-process implementations are identified, the information is fed back into the respective level specification model, and the faults adjusted early on. In this way, faults can consistently be detected and rework cycles reduced or even avoided. In general terms, the practice of not basing the implementations on foundational theories can result into an increased number of flaws. Thus, in order to implement high quality functionality from scratch, the TMAPE path must be followed.
In some practical cases, the existing methods, algorithms or codes that can approximately solve problems are sometimes available, and can be adapted and used. For instance, if there is an implementation that roughly matches the expected functionality, a backward-forward path i.e. PAMT-TMAPE can be followed to scale the existing codes to the problem. Similarly, if there is an algorithm that closely matches the specifications of the problem, a backward path, for this case AMT, can be traced, followed by a forward transformation of TMAPE, and so on. The challenge that the developers may face when reusing the readily available in-process implementations is how to interface them in an effective way and how to synchronize the needed modification to the existing implementations. One of the dangers of following the reuse paths is that even mediocre concepts can be institutionalized.
In conclusion, the principles of AP can be summarized as follows:

- The software design process passes through various stages, implicating diverse in-process abstract appearances of software in different contexts.
- Theories, methods, algorithms and pilot prototypes are testable in-process implementations at the design phase of the software development process. We refer to the corresponding creation stages as *levels of abstraction*.
- Requirements are clustered according to the levels of abstraction, and
- The representatives of the stakeholders are systematically involved in the assessment of quality of the in-process implementations as they evolve. The evaluation criteria are derived from the requirements. The requirements are significantly improved through the elaboration processes that involve evaluation of the representations of the in-process implementations, also called *abstract prototypes*.

2.1.2. The Process

AP is essentially the symbiosis of a methodology and software tools usage in the representation and review of preliminary implementations of CAD/CAM/CAE software. It furnishes the developers with a methodology for exploration and reasoning about the alternative solutions during conceptualization of functionality and systematically brings into the review floor various stakeholders of the envisaged software, namely various user groups, developers and other experts. It thus helps developers think far beyond their own experience and expertise and reach across stakeholders and other experts, to find solution to problems. It can also be regarded as a way of thinking that enables the developers to project and reflect the contents of their solution.
Fig. 2 shows the main activities in AP. AP comprises a Meta scheme that shapes the AP process among the levels of abstraction. It also consists of specific schemes for (i) shaping and directing activities within the individual levels, (ii) clustering requirements according to the levels of abstraction, (iii) finding weak spots in the in-process implementations and ranking of alternative solutions, and (iv) guiding the participation of the stakeholders in AP. These schemes are presented in detail in [14].

2.2. Software Support

The AP software has been developed based on the principles of AP covered in [14]. The AP software portfolio includes utilities for representation and processing of requirements, representation and processing of abstract prototypes, preparation of opinions gathering tool, and information analysis. Computers bring into AP the power to communicate, store and process information. The essence of AP being aided by computers is the marriage created by applying its underlying schemes and the strengths of computers to provide assistance to the developers or enhance their capabilities. The intention is to make the AP procedure highly knowledge intensive by including the requirements as well as knowledge about the problems and solutions in the AP software. This makes AP more effective in supporting software development process.

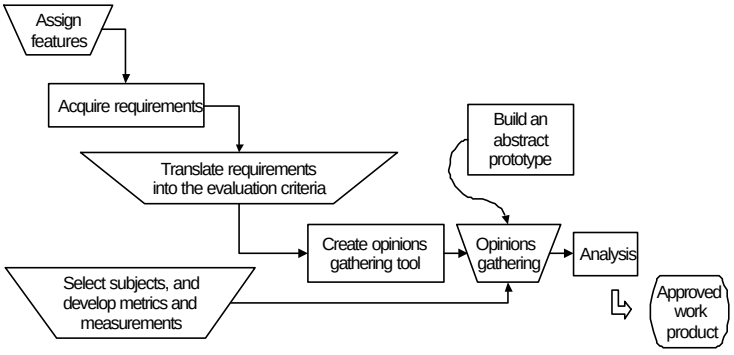

Fig. 2. *The AP process*

Table 1. *Paths followed in the development of the studied software tools*

Software tool	Paths followed	Result
Speech input utility	EPAMT/ TMAPE	Requirements fulfilled sufficiently
Vague geometry modeler	TMAPE	Faster and efficient
Mechanical behavior simulator	TAPE	Flexible as required
Photogrammetry	TMAPE	Many requirements taken into consideration
3D points manipulation UI	TME	Worked as desired

At the initial stages of the AP process, the role of the AP software is to support off-line preparation activities. These include (i) supporting acquisition of requirements, (ii) providing expert guidance, for instance, during the selection of the forms of representation of abstract prototypes or the subjects, and (iii) availing the AP knowledge and guidelines. In the late stages of the AP process, the AP software supports processing of field information. This yields results, which can be used in the identification of flaws and selection of the best alternative(s). The AP system also allows for requirements, abstract prototypes, and the analysis results to be stored for use during the development interval as well as in future projects. Also, the AP system serves as an online mentor that makes process practical by providing extensive guidelines, templates, and examples. It is important to emphasize that goal of the AP software has not been to achieve full automation of the AP process, but rather to provide software tools for assisting the developers in AP. Human interaction and supervision is always of enormous importance in AP. Detailed description of the AP software including its algorithms and utilities are available in [15].

3. CASE STUDIES

This section summarizes the results of case studies on application of the AP concept. The case project was on development of software tools for supporting engineering design, namely shape conceptualization and reengineering of shape. Table 1 shows the case software tools, of which the design processes were studied, paths followed during their design, as well as the opinions given on the software products that resulted from following the indicated paths. As can be seen, different paths were followed, and this largely depended upon what kinds of solutions were known at the beginning of the design process. Nevertheless, ultimately in all cases the developers felt that the end

products were developed as specified in the requirements model.

The quasi-automated procedure depicted in Fig. 2 was followed when reviewing theories, methods, algorithms or pilot prototypes. Many design phase in-process implementations have been reviewed, but due to space limitations we only explain, using examples, how the reviews were conducted. Detailed descriptions of the AP reviews can be found elsewhere, for example in [16]. When an in-process implementation had to be reviewed, its requirements were formulated; starting point being the pool of previously acquired requirements (at the preceding phase of requirements specification) stored in the AP system's database. The requirements were then made specific to the current stage of the design process and eventually transformed into evaluation criteria (that is, paraphrased into a language understandable to the targeted subjects). Fig. 3 shows typical examples of the evaluation criteria used in various levels of abstraction.

In each AP exercise, an information-gathering tool comprising of the evaluation criteria [Fig. 3] was prepared and sent out to the subjects representing the targeted stakeholders for completion. The opinions were subsequently analyzed and the levels of fulfillment of the evaluation criteria (that is by default the levels of fulfillment of the requirements[1]) as well as the acceptability of the in-process implementation in question determined.

Identifier	The evaluation criterion	Relevance# - ρ (0…3)	Score* - ϕ (0…3)	Confidence! - χ (0…3)
R1	Each action in the sequence has a successor			
R2	Steps of an algorithm precisely are defined			
R3	The algorithm terminates after a finite (required) number of steps.			
R4	The outcomes of each stage are known			
R5	The sequence of actions has a unique initial action.			

0 - Not Relevant; 1 - Fairy Relevant; 2 - Relevant; 3 - Very Relevant
* 0 - Not Fulfilled; 1 - Fairy Fulfilled; 2 - Fulfilled; 3 - Very Much Fulfilled
! 0 Not Confident at All; 1 - Fairy Confident; 2 - Confident; 3 - Very Confident

Fig. 3. *A typical information-gathering tool*

The information gathering and analysis process can be described as follows. Subjects are required to specify the level of fulfillment of each requirement (ϕ), relevance of every requirement (ρ), and their confidence (c) [Fig. 4] on the information gathering form. The specified values depict how the subjects feel requirements have been fulfilled, are relevant, and how accurate they responded respectively. For m review criteria used in an AP exercise in which n subjects participated, the total merit value m_T for a solution proposal can be determined as follows:

$$m_T = v_i \sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij}\, r_{ij}\, c_{ij}$$

where ϕ_{ij}, ρ_{ij}, and χ_{ij} are the level of fulfillment, the relevance, and the confidence about requirement i as expressed by subject j. and ω_i is the weight assigned by the requirements engineer (the developer), which signifies the importance of the requirement i. The acceptability index (α) is then defined as follows:

$$a = \frac{m_T}{e_s}$$

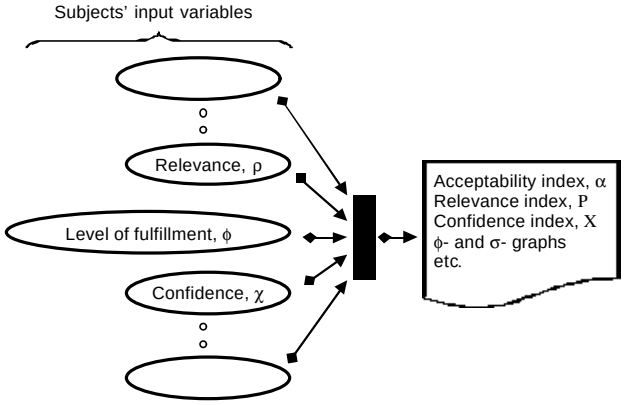

Fig. 4. *The evaluation scheme*

where, $e_s = nm(f_{\max} r_{\max} c_{\max})$ is the maximum possible total merit value. ϕ_{max}, ρ_{max}, and χ_{max} are the maximum achievable values of ϕ, ρ and χ respectively. α gives the developers clue on the extent to which the proposed solution compares to the ideal solution. Based on the values of a, the solution proposals can be ranked in pecking order of salience. m_T values for requirements can also be presented graphically i.e. used in plotting the σ – graph, while the collated ϕ values can be used in the generation of the ϕ – diagram [see Fig. 5]. These graphs provide pictorial overview of the extents of fulfillment of requirements. The ρ and χ values are also used in the determination of the *relevance* (P) and *confidence* (X) indexes respectively as follows.

$$\mathrm{P} = \frac{\sum_{i=1}^{n} r_i}{n r_{\max}}$$

$$\mathrm{X} = \frac{\sum_{i=1}^{n} c_i}{n c_{\max}}$$

The P index indicates how relevant, in the opinions of the subjects, the review criterion and

[1] Each evaluation criterion consists of a tag called *identifier*, which is used for cross-referencing the evaluation criterion to the requirement it relates to. Based on this, the evaluation criteria can be correlated to the requirements.

consequently the requirement is, in regard to the ongoing AP exercise, while the X index indicates how confident the subjects have assigned ϕ and ρ values, and how knowledgeable they are. Refer to [15] for further elaboration on the information gathering and analysis process.

Fig. 5 shows a typical plot of the levels of fulfillment of requirements. Further descriptions on how to prepare these graphs are well as on how to determine the acceptability, relevance and confidence indexes for in-process implementations are documented elsewhere, for example in [15-17]. These plots show relative levels of fulfillment of requirements by the in-process implementation, that is, to what extent the in-process implementation in question satisfy requirements. Based on these diagrams, the requirements that have not been fulfilled can clearly be distinguished, and the in-process implementation improved while keeping an eye on the least fulfilled requirements.

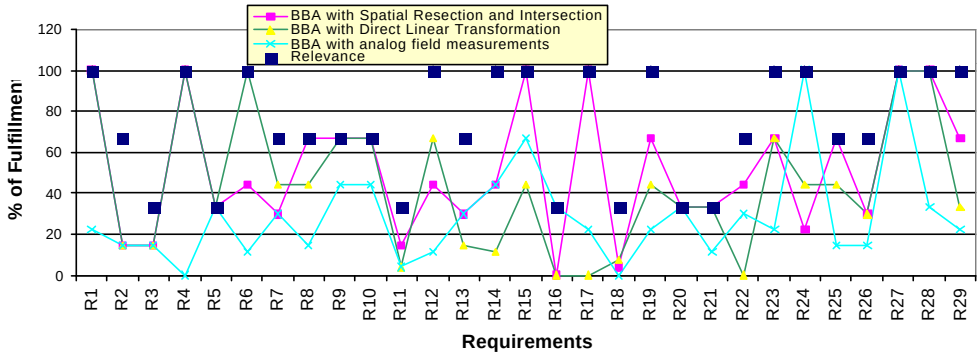

Fig. 5: *A typical plot of the levels of fulfillment of requirements*

The lessons learned from case studies can be summarized as follows.

- In all application cases, the Theory-Methods-Algorithms-Pilot prototypes (TMAP) paradigm was intuitively followed, but not necessarily in a strict singular order. The tracks pursued largely depended on in what form the solution was initially available.
- Comprehensive understanding of foundational engineering principles was always necessary when designing CAD/CAM/CAE software. This was always the case, even for problems that had high-fidelity initial solutions. Algorithms or codes for the building blocks of CAD/CAM/CAE software were adapted and used only after thorough investigation, and even after modifying underlying theories or methods.
- Involvement of clients in the reviews at the 'methods' abstraction level, especially in highly technical and scientific CAD/CAM/CAE software projects, was seen by some developers as rather unfeasible. This was because introducing highly technical and scientific concepts to clients required a lot of efforts, and it was difficult to make them understand such concepts as the developers wished. For such projects, involvement of the clients in AP had to be delayed until after implementation of the pilot prototypes.
- Requirements for in-process implementations were highly transferable from task to task, and project to project. In most cases, the peer level general-purpose requirements were used unchanged in various AP experiments. The research-oriented case studies were conducted first, and the overwhelming numbers of the general-purpose requirements for the in-process implementations were used unchanged in the industrial case study.
- The application cases have attested that the AP methodology can effectively help CAD/CAM/CAE software developers at the design phase to (i) work in an orderly and guided manner, (ii) quickly acquire requirements for the in-process implementations, and (iii) systematically review the in-process implementations, by involving various CAD/CAM/CAE software stakeholders as the evaluation subjects. It has been observed that the AP procedure provides a framework for determination of adequacy of the in-process implementations at the design phase and helps in the identification of weak spots.
- There have been skepticisms on the reliability of AP metrics and measurements. In measuring the acceptability of the in-process implementations or the extent to which they fulfill requirements, some users questioned the objectivity of the collected data, and pointed to composition of the review panels and completeness of review criteria as the reasons behind their skepticism[2].
- Some of the developers who used the AP methodology and software tools could not immediately understand what a theory or a method means, or differentiates theories from

[2] In general terms, skepticism on metrics and measurements is common in many software development projects or organizations. Nevertheless, it is understood that without metrics it is impossible to know how well various products have been implemented.

methods straight away. Often it was necessary to provide definitions. Perhaps this is due to the fact that theories and methods are not commonly recognized as intermediate in-process implementations in the existing software development models. Products of the later CAD/CAM/CAE software design stages (namely, algorithms and pilot prototypes) were, however, easily distinguishable.

- The AP concept appeared to work slightly more convincingly in the industrial CAD/CAM/CAE software project than in the research oriented CAD/CAM/CAE software project. This can be attributed to the nature of the projects. In the industrial CAD/CAM/CAE software project, the needs and the requirements were much clearer and testable, while in the research oriented CAD/CAM/CAE software project, the needs and the requirements were somewhat vague and sometimes some of them were rather difficult to test. Nevertheless, it can be said that the philosophy of AP was evenly appreciated in research oriented as well as business oriented CAD/CAM/CAE software project.

A questionnaire survey was carried out to investigate the acceptability of the AP concept. The developers and various subjects involved in the AP reviews were asked to indicate how effective AP helped or support various aspects. A rating scale of *0 to 3* for *not effective, fairly effective, effective, and very effective* respectively was adopted. Statistics shows emphatically that the respondents believed that it is a useful strategy. All respondents indicated that it is effective (80%) or very effective (20%) in shaping and directing activities at the design phase of the software development processes while over 80% felt that it is effective or very effective as a pre-implementation testing strategy. Over 50% of the respondents indicated that it effectively or very effectively helps identification of weak spots in the design phase in-process implementations.

4. CONCLUDING REMARKS AND FUTURE WORK

A new strategy to reduce the number of faults in the design phase of the software development process has been developed. The AP strategy has in the first place been designed as a general procedure for pre-implementation testing of any CAD/CAM/CAE software. The idea is to introduce review cycles early on and interweave them in the design process. This can ultimately help reduce flaws. The application case studies presented in the previous section illustrate two ways of using the AP strategy, namely, (i) as a model for directing the development and review activities at the design phase; and (ii) as a technique for keeping requirements and constraints aboard when designing CAD/CAM/CAE software. There can, however, be other application avenues. For instance, it can be used as a technique for systemization of the involvement of various stakeholders and as a framework for progressively introducing methods and software into the industry. Bringing into the review floors the representatives of the future users to serve as members of the review panels manifests the later application orientation.

The application case studies have shown that AP effectively supports the developers in exploring the suitability of the design phase in-process implementations. It has been demonstrated that it (i) offers a platform for shaping and directing software design activities and for systemization of the involvement of various stakeholders in checking progressively if requirements are being satisfied when designing software; (ii) provides means for identification of weak spots in the in-process implementations and measuring how well they have been developed, thus offering means for early recognition of the needs for enhancement; (iii) enables investigation and consideration of the aims, possibilities, and various aspects in a systematic way early on, (thus, bad decisions that could otherwise jeopardize software project can therefore not be institutionalized), and (vii) provides a systematic way for selecting and enhancing solutions. It can be said that AP reduces the imperfections that often lead to rework of poorly engineered software as well as the susceptibility of the software design processes to errors. As a result of application of this technique, only proofed and reliable theories, methods and algorithms can be institutionalized. The application of this strategy can ultimately warrant creation of usable and low risk software.

What is perhaps most different is the structured levels-wise prototyping and reviews of the design phase in-process implementations and systematic involvement of various stakeholders in this. Giving them chance to express their opinions about the implementations helps reduce the risk of developing substandard CAD/CAM/CAE software. For the AP methodology to be more successful, it needs to be integrated with suitable metrics, an appropriate software process model, and complemented with traditional verification and validation techniques. It is; however, fair to mention that the AP methodology has not yet been tested in many practical situations. Further modest enhancements may certainly be required, and more tests need to be conducted before it is put in use in real world situations.

In spite of the contributions of this work, still there are open research issues to address in order to achieve the goal of having a more effective methodology for pre-implementation testing of CAD/CAM/CAE software tools, or extending the usability of the achieved results. To enhance the AP concept and to widen its application domain, further research needs to be carried out in various directions, for instance, to:

- Explore how the AP concept can best be used alongside the major traditional software process management strategies.
- Investigate how the AP concept can effectively be used in conjunction with conventional software V&V strategies.
- Study and establish the consequences of usage of the AP technique on time to market and development costs.
- Further enhance the AP concept and investigate the possibility of extending its application domain to include other similar software products.
- Extend the scope of the AP technique beyond software products to include, for instance, pre-implementation testing of artifactual products.

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

References

[1] Glass, R.: *Building Quality Software*. Upper Saddle River, Prentices Hall, 1992.

[2] Melhart, B.: *Software Engineering*. In Encyclopedia of Computer Science, Fourth Edition, Nature Publishing Group, London UK. 2000, pp. 1606-1611.

[3] London, R. L. and Craigen, D.: *Program Verification*. In Encyclopedia of Computer Science, Fourth Edition, Nature Publishing Group, London UK, 2000, pp. 1458-1461.

[4] Petschenik, N. H.: *Building Awareness of System Testing Issues*. In Proceedings of the 8th International Conference on Software Engineering, 1985, pp. 182-188.

[5] Brooks, F.: Mythical Man Month: Essays on Software Engineering. Reading, MA: Addison-Wesley, 1995.

[6] Brooks, F. P.: No Silver Bullet: Essence and Accidents of Software Engineering. Computer, No. 20. Vol. 4, 1987, pp. 10-19.

[7] Howard, A.: *Software Engineering Project Management*. Com. of ACM, Vol. 44 No. 5, 2001, pp. 23-24.

[8] Armour, P. G.: *Software as a Currency*. Com. of ACM, Vol.44, No. 3. 2001, pp. 13-14.

[9] Lieberman, H. and Fry, C.: *Will Software Ever Work*. Com. of ACM, Vol.44, No. 3. 2001. pp. 122-124.

[10] Eick, S. G., Loader, C. R., Long, M. D., Votta, L., G. and Wiel, S., V.: *Estimating Software Fault Content Before Coding*. ACM 1992, pp. 59-65.

[11] Humphery, W. S.: *Managing the Software Process*. Reading, Massachusetts: Addison-Wesley, pp. 353, 1989.

[12] Lewis, E., W.: *Software Testing and Continuous Quality Improvement*. 2nd Edition, CRC Press, 2000.

[13] Jones, W. G.: *Software Engineering*. John Wiley & Sons, 1990.

[14] Opiyo, E. Z., Horváth, and Vergeest, J. S. M.: *Abstract Prototyping of Design Support Tools: Methodology and Preliminary Results*. Proceedings of the 25th Design Automation Conference (DAC), Las Vegas, Nevada, USA, Paper No. DETC/CIE-8551, 1999.

[15] Opiyo, E. Z. "*Facilitating the Development of Design Support Software by Abstract Prototyping*" Ph.D. Thesis, Delft University of Technology, In press.

[16] 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, September 10-13, Baltimore, MA. USA, Paper No. DETC/CIE-14613, 2000.

[17] Opiyo, E. Z., Horváth, and Vergeest, J. S. M.: *Using the Abstract Prototyping in the Development of Design Support Systems*. Proceedings of the 21st Computers and Information in Engineering (CIE) Conference: September 9-12, Pittsburgh, PA, USA, Paper No. DETC/CIE-2233; 2001.