ABSTRACT PROTOTYPING IN SOFTWARE ENGINEERING: A REVIEW OF APPROACHES

Els Du Bois[1], Imre Horváth[2]
(1) Artesis, (2) Delft University of Technology

ABSTRACT
Abstract prototyping (AP) is a pre-implementation testing approach in software engineering, based on low-fidelity prototypes. It supports demonstration and evolution of software concepts at an early stage. It allows designers to optimize the operation of the software and allows end users to understand how to work with the system. In this paper we survey various 'approaches', i.e. both the way of developing the content of AP and the manner of using them in software engineering. We developed a reasoning model intuitively and defined research questions to structure our review and this paper. Our objective was to get insights in the existing definitions, information contents, construction processes and application opportunities for AP. We have found that AP is simultaneously a challenging scientific and a complex practical issue, which usually raises a large number of sub-issues and questions. In addition we observed that there are multiple interpretations of AP which are disturbing a clear picture. Based on the findings we observed that it is possible to generalize the key constituents of AP and to integrate them into a simplified and application-independent AP methodology. This methodology and its applications have been published in other publications.

Keywords: abstract prototyping, design support tool, early stage, code-free pre-implementation testing, low-fidelity prototyping

1 INTRODUCTION
Abstract prototyping (AP) is a common term to describe various approaches of developing early replicas of software tools. Below, and also in the title of this paper, we use the word 'approach' to refer to the way of developing contents and the manner of using abstract prototyping in software engineering. In this paper we will survey the current state of the art and analyze general and specific approaches with the objective of concluding about a general framework of AP. The intention of our background research is actually to develop a rapid abstract prototyping approach, which allows us defining design support software with the involvement of the end users and various other stakeholders. We are especially interested in approaches that give preference to rapidness and convenience of AP development, rather than to the issue of achieving high fidelity and comprehensiveness.

Multiple papers are mentioning that many of the faults detected in existing software can be traced back to the problems of requirements specification, pre-implementation testing and user conformant evaluations [1-3]. The problem is that the activities in different phases of software development, in particular in the design phase, do not scale to precisely match the underlying needs of the users [4]. It has also been recognized that the involvement of representative software users in the development process is valuable, because it significantly improves the acceptance of the final product. Ultimately, usability comes from fitting the architecture and the content of the user interface to what the users are trying to accomplish [5]. Obviously, it is more costly to conceptualize something incorrectly and then to sort out the problems. Consequently it is cheaper to design and build the software right at the first time and to reveal all unforeseeable problems in an early phase. Usually a systematic approach is needed, because software errors are typically deeply and intimately embedded in the architecture of the software. In addition to the functional errors, the user and usability aspects should also be taken into consideration. The user aspects can be taken into consideration in software development by aggregating knowledge about the future users or by directly involving them in a participatory software development. This form of software co-development has been named participatory design [6-8].

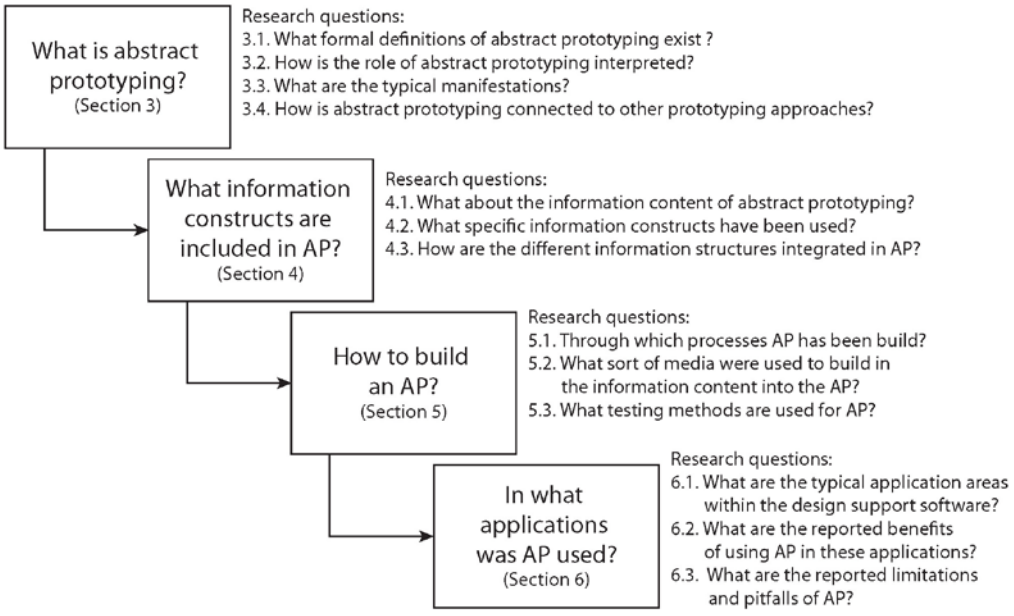
In most cases, participatory design is mainly done with the involvement of limited number of potential users and low-fidelity prototypes, which are easy for the users to get familiarized with and which they

can learn and employ by themselves. Since end users have neither knowledge nor experience in reading source-codes of software tools, designers need to use efficient communication means. In addition to allowing hands-on experimentation, it is crucial that the demonstrative software includes more than just the interface. By providing the end users with clues about the structure and content of the developed software, we can support their understanding of the possibilities and limitations of the system, well beyond what is implied by the interface [4]. For the reason that typically a lot of information is still missing, one of the major challenges of AP is that only low-fidelity ones can be developed at the beginning of the development process. The development of high fidelity prototypes would assume a much wider pool of information, which is only available in the later detailing phases of software development. However, in this case, the involvement of other software experts and end users would be needed to support software conceptualization and exploration of errors [9, 10].

According to our knowledge the use of AP is widespread and there are multiple interpretations, which are somewhat disturbing the formation of a clear picture on the whole. On the other hand, AP is a challenging scientific as well as a complex practical issue, which may raise a large number of sub-issues and questions. In order to have a systematic overview of the approaches, the second chapter will introduce a reasoning model, which has been used in our study. In the next sections, we are addressing four major issues associated with AP. In Section 3, we consider the currently existing definitions of AP and analyze their differing manifestations. In Section 4 we investigate the information content of abstract prototypes, as well as the constructs and models that have been proposed to develop these abstract prototypes. Section 5 deals with the development processes and the developmental resources of AP. In Section 6 we concentrate on the applications of AP and investigate its benefits and pitfalls. In Section 7 we introduce a generic definition for AP and define the main constituents, which enable us to operationalize the AP methodology in the development of design support tools that represent a specific domain of software engineering applications.

2 EXPLANATION ON THE APPLIED REASONING MODEL

Over the years, AP has developed to be a complex model from information technology, business communication and design methodological points of view. The issues related to the definition, implementation and employment of AP in user-centered design have been discussed in different context. Our forerunning investigations explored that what exist at this moment is featured by the lack of a set of consolidated application independent definitions, the vague use of the concepts in practical implementations and the different objectives, which are set forth in software, artifact and surveys development projects. Having recognized this situation and striving after a good level of rigor, it became indispensable to underpin our discussion of the phenomenon of AP in the context of designing support software by a logical reasoning model.



The major objective of our literature study is to gain insights in the existing definitions, information contents, construction processes, and application opportunities of AP. Without any further explanation, it can be seen that these four aspects create a kind of logical flow for the study and hence, this has been incorporated in the reasoning model, as shown in Figure 1. Concerning each aspect of the study, research questions have been derived, which orientate the investigations towards technical details. A representative set of these research questions is included in the graphical representation of the reasoning model. The importance of this reasoning model is underpinned by the fact that the papers published so far, related to content development, structuring and applications of AP, only partially or not at all addressed this range of research questions.

We have to mention that this reasoning model has crystallized out in an iterative process of studying the literature. Our initial concept has been modified as we aggregated information about the different approaches and issues raised by other researches. It has to be noted that even the final version of the reasoning model cannot capture the content of all studied papers and has a kind of demarcating flavor. It means that the answers to the specific research questions could just be partially found in the individual papers, and sometimes, we had to combine the explanations of multiple papers to arrive at a meaningful answer, or we had to reuse the discussion in a single paper to argue about multiple research questions. We found that the major limitation of the reasoning model is that it was not able to create integrity of all studied aspects. We had to overcome this lack of integrity in the presentation of the findings by interrelating them in the discussion part of the paper. As a forerunning concluding remark we can claim that this reasoning model proved to be useful to investigate the current state of the art in the view of the literature.

3 WHAT IS ABSTRACT PROTOTYPING?

3.1. What formal definitions of abstract prototyping exist?

In the literature a large number of synonyms have been used to identify abstract prototyping such as pre-implementation prototyping, low-fidelity prototyping, automatic prototyping, rapid prototyping, early prototyping, low-cost prototyping, surrogate modeling, media prototyping, pre-implementation testing, paper prototyping etc. For most of these notional terms also definitions have been provided. However our impression is that these definitions are partially overlapping and none of them can be accepted as absolutely complete. The simple reason is that these definitions reflect different stances, express different viewpoints and objectives and have been proposed for different purposes. Since it is impossible to individually analyze all of them, below we cite only the most representative ones:

'AP is a pre-implementation testing methodology, it can effectively help developers to (i) quickly elicit and formulate requirements, and (ii) systematically test abstract software implementations, by systematically involving various stakeholders as subjects.' [11]

'Lo-fi prototyping is the visualization of design ideas at very early stages of the design process.' [12]

'An abstract prototype allows designers to describe the contents and overall organization of a user interface without specifying its detailed appearance or behavior.' [2]

'Rapid prototyping of interactive systems is a technique used in order to assess design ideas at early stages of the development process. It attempts to foster the collaboration between all the stakeholders involved in the project and to facilitate iterative cycles of reviewing and testing.' [13]

'In rapid prototyping of a user interface, design ideas are tested out on potential users with a prototype that is relatively quick and inexpensive to construct.' [14]

The above selection of the definitions clearly shows the commodities of the various interpretations. More important for us is to see that there are differences in the extent and coverage of the process of AP. In some cases it is supposed to cover the whole process, in other cases it is assumed that only some specific activities are supported by AP, more specifically the visualization of the software. It implies that we have to strictly differentiate the concept and meaning of abstract prototype from the procedure of abstract prototyping. Prototypes are representatives of design ideas, which may manifest in multiple forms. Prototyping refers not so much to the activity of making abstract models and representations but to utilizing prototypes in the software development process [10]. A comprehensive

view on AP however can consider both the process and methodology of making abstract prototypes and the use of abstract prototypes to generate additional information for the development process.

3.2. How is the role of abstract prototyping interpreted?

We have to start out of the fact that, as it is also indicated by the above definitions, AP is used for various purposes in various contexts. The common objectives of use are (1) aggregation of information, which cannot be obtained otherwise (i.e. without developing prototypes), (2) to attain a comprehensive image on the operation and interaction possibilities, and (3) to formalize the information inquiry and the whole of the software development process.

With regards to these objectives, the major roles what abstract prototyping can play in software development have been identified by researchers as follow. Namely, AP: (1) facilitates the communication to the end users in the early phase and the adaptation of the software-in-development to the user needs [8]; (2) makes the ideas tangible for the developers themselves [10]; (3) assists in clarifying the interface of the system [15]; (4) helps to identify the functional boundaries of the system [16]; (5) facilitates making a forecasting on the needed resources [17], supports making estimations on the needed system development capacities, money , time, infrastructure, etc. ; (6) supports the exploration of errors and reduces the potential pitfalls [18]; and (7) gives means of process monitoring, and of systematizing the process [3].

3.3. What are the typical manifestations?

In the literature, there is a distinction between formal and informal specifications of abstract prototypes. These specifications are usually referred to as informal models, but they are also called mental simulations. Informal models are seen as emergent models, which are not documented precisely and just used to communicate and explain design ideas [3]. They can we sub-divided into mental models and paper-based models. The creation of mental model happens with the goal to raise a shared awareness of the design concepts, while a paper-based prototyping intends to achieve a shared awareness through explanations presented typically in graphical forms without any specific supporting method or structure. These represent one branch of model manifestations as shown in Figure 2.

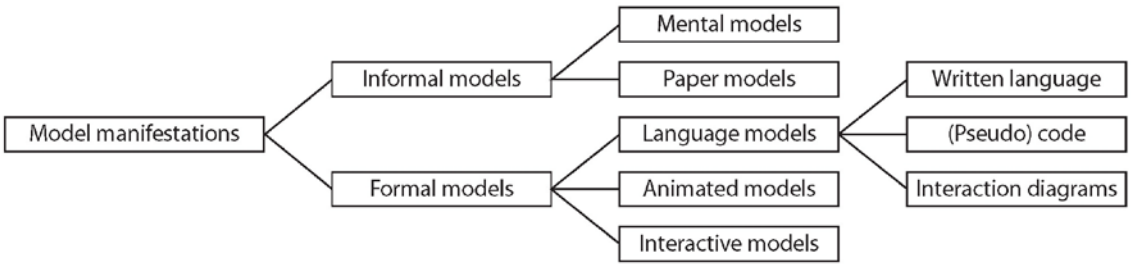


Figure 2: Overview of the typical manifestations

In the literature there have been several semi-formal and formal models presented, ranging from symbolic representations through functional representations to language representations. Formal models are developed to create a structured manifestation of the abstract prototype. They can be further decomposed into language-based models, animated models and interactive model. Language-based models are demonstrative specifications created by using (i) common verbal or written language [19], (ii) interaction diagrams and pseudo-codes, such as canonical prototyping [20], and (iii) specific languages, such as UML or HTML [21]. Animated scenario models are showing the operation of the software system but without the opportunity of introducing immediate changes in the software concept [12]. Interactive formal models allow communicating with a semi-running program, often relying on the Wizard of Oz simulation [17].

3.4. How is abstract prototyping connected to other prototyping approaches?

For the reason that physical prototyping and virtual prototyping play no such a role in software engineering as they play in the prototyping and testing of artifacts, we consider only the relationship of low-fidelity AP to high fidelity prototyping in software development. A fundamental relationship between low-fidelity and high-fidelity prototyping techniques is hiding in the coupling of their information contents. Typically abstract prototypes capture descriptive information on a higher level of abstraction than high-fidelity prototypes. Therefore, there is a need for information conversion or

transformation, which requires human interaction. This is somewhat similar to the relationships of technical drawings and sketches to digital virtual prototypes in the development of artifactual products. A second issue is the completeness of the information contents. In case of abstract prototyping, the goal is to represent the in-process product with a minimal amount of information in order to save time on the side of the designers and to reduce the cognitive load on the side of the end users. However high-fidelity prototypes should capture the largest amount of descriptive information in order to provide sufficient intelligence for a comprehensive assessment of solutions, and for elimination of the potential errors. On the current level of semantic processing of information, neither the transformation nor the completion problem can be addressed in an algorithmic way, i.e. human involvement is needed to complete these information conversion tasks [22]. This situation cannot be changed easily because AP can do its best in the requirements specification and the conceptualization phases, while the high-fidelity prototyping approaches cannot be omitted from the detailed design and verification phases [10, 12].

4 WHAT INFORMATION CONSTRUCTS ARE INCLUDED IN ABSTRACT PROTOTYPING?

4.1. What about the information content of abstract prototypes?

As explained in sub-sections 3.1 – 3.3, abstract prototypes are developed for different objectives, play different roles in the software development process and are represented through the use of discrete partial models. What is implied by practically all definitions is that an abstract prototypes is actually a combination of purposeful knowledge packages [23]. These knowledge packages are capturing and recording information about (1) the actors and the stakeholders on the human side, (2) the system functionality and interface on the system side, and (3) the context of application which interconnects the end users and the system. These fundamental knowledge packages are arranged graphically in figure 3. In more detail, the knowledge packages should contain sufficient amount of information to describe the actors, the stakeholders, the system interface, the interaction procedure, the system functionalities, and the system controller, as well as the context of application and the stakeholders' objectives.

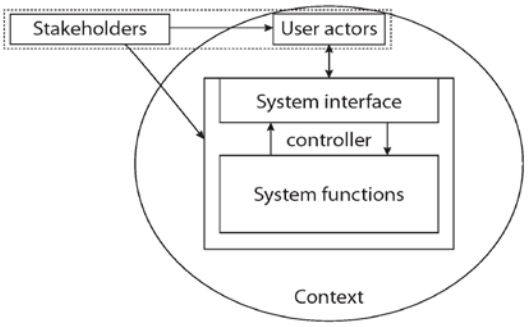


Figure 3: Fundamental knowledge packages for building AP

Different authors presented different interpretations on the content of implemented abstract prototypes. According to Haesen, the abstract prototype should convey information about aspects such as usability requirements, scenarios, and personas representing the user needs [22]. Brandt argued that it is necessary to learn about users and the context of use, and to create a common understanding of the development task through the content of the abstract prototype [7]. Weidenhaupt highlighted the benefits of creating concrete, use-oriented system description prior to modeling the functions, data and behavior [24].

4.2. What information constructs have been used?

The literature seems to agree on that the above knowledge packages should all be represented in an abstract prototype. To facilitate this, various information constructs have been defined. An information construct is a formal information structure, which provides information on the components of the knowledge packages. The information constructs are underpinned by reasoning models and represented by language means. Model-based specification of the construct has gained popularity in software development because models can document the result of discussions and communicate the

understanding of designers on the functional and usability issues [2]. In the following paragraphs, we discuss these information constructs according to their belonging knowledge packages.

Modeling the users (as actors) and the stakeholders was addressed in many papers. Constantine made an overview of the approaches of specification of the actors and the stakeholders [25]: The user actors, as well as the stakeholders have been represented as typified personas. Other papers gave preferences to circumscribe the actors through user roles, rather than as personas. The user roles, which specify the relationships between some users and the software, is described by a user role card containing information about context, criteria and user characteristics. Actors and their user roles are shown in a user role map. This information construct is called user role model or participation model [26]. In the context of interface design, Constantine also proposed an essential modeling technique for modeling the system in the field of software engineering [2, 5, 21, 25-28]. This is based on task cases, which are activities that should be accomplished concerning both the system functions and the actions of the users in order to achieve the goal of using the system. Often the term 'use case' is also applied, but in this cases more concrete and detailed information is given on the activities [15]. Later on use cases can be transformed into a sequence of primitive actions [29]. Some authors have also discussed task scenarios and user scenarios [30, 31]. The task cases are expressed in a task case map or in a use case map [16, 32]. All these pieces of information come together in the task model or in the performance model. In addition, they are also considered together with the user role model in a content model, which also called workflow model or system requirements model [29].

Li et al. and also Constantine argued that besides the task model, there is a need also for a conceptual class model or domain model. According to Constantine another required model is the navigation map, which gives additional information on the system interface. It consists of labeled symbols representing interaction contexts and lines representing transitions among them. Furthermore, information about the context of using the software is also needed to be built in the AP, and this is typically achieved by the activity map and operational model of the context. As elements of the interface model, tools and containers had to be defined. A content model is an abstract representation of the tools and containers that need to be presented to the user in particular use cases. The above core models are developed in association with other models and are interlinked with them. One example is the domain model that embodies the underlying logic and constraints of the application, and another one is the operational model that captures salient aspects of the working environment or operation context. Though it is necessary to integrate the information constructs in order to get to a comprehensive abstract prototype, this is not so straight forward in the practice due to their varied information contents. This explains why just a limited number of elements have been interconnected in the proposed approaches so far.

4.3. How are the different information structures integrated in AP?

If we want to construct a generic abstract prototyping model from the models proposed in Section 4.2, we have a mess of information, without any logic or cohesion. Therefore we have to impose a structure on this generic model, which can be built either as an aggregative model, or as a decomposition model. Aggregative model building can be seen as a bottom-up approach, in which the different aspect models are combined under one umbrella [33]. The problem with the aggregative approach is that the translation of all different sub-models into one model is hard because they are typically built with different objectives and resources [34]. In addition it is difficult to determine how the different sub-models should interoperate in order to depict a holistic procedure for the development and application of the AP. The technical difficulty originates in the fact that sub-models cannot directly interoperate based on different schemes.

An alternative is to use a top-down approach. This would start out from a meta-model, or system model, and derive the information content for the various sub-models by specializing the content of the meta-model on multiple levels [35, 36]. The problem of the top-down approach is the complexity and the comprehensiveness that is needed to cover everything. As indicated, both approaches raise various implementation issues. It looks like there is no one universal solution due to the fact that both approaches have their own advantages and disadvantages, and more importantly, there are two communities behind who give preference to one or another approach. We think that the best approach is to combine the bottom-up and top-down approaches, which needs a dedicated methodology. This conclusion has also been confirmed by Palanque [37].

5 HOW TO BUILD AN ABSTRACT PROTOTYPE?

5.1. Through which processes AP has been build?

In the studied literature we found many authors who proposed specific processes for AP. For instance, Rettig defined a logical process for doing paper prototyping [9]. Snyder defined a chronological process for paper-based prototyping [17]. Brandt proposed to facilitate the development of the abstract prototype by means of design games [7]. Dijk developed a process to test shape manipulation tools by using abstract prototypes [38] . What we could observe was that all these papers proposed to use the same logical process of AP. This process in general consists of two main phases: (1) building the abstract prototype, and (2) testing and evaluating the abstract prototype. It is also common that the AP is build based on the information content and structure discussed in Section 4. So the only differences we could find were in the methodology used to perform these phases, and in the chronology of the execution of the actions. (We note that a different chronology is presented in reference [17] than in other references). In the phase of building the abstract prototype, all necessary knowledge content is collected first, then the information is compiled into a scenario of the happenings and communicated in the form of narration [19]. The second phase of the abstract prototyping process starts with the preparation for the testing in which the experiment is designed and pre-tested. Demonstration and assessment of the AP is the focus of the experiment. In the assessment step, the stakeholders' opinions and change proposals are sorted, analyzed, ranked and validated. Testing of AP is a research issue on its own right and hence further information about the method is given in Section 5.3.

The narration presents the story to be delivered by the abstract prototype in addition to explaining the system functionalities and the human interaction and behavior. The narration offers the opportunity for the stakeholders to interpret, infer meanings that are not explicit, and further think about the information communicated. It also gives them a lived experience because the stakeholders can feel that they are taking part in the presented operation and interaction processes [39]. The stakeholders can be involved in the process of AP in different ways. For instance, they can passively process the narration and make conclusion based on that or can play an active role through interaction with the abstract prototype as a test person [23]. In order to increase the richness of information and the efficiency of processing the abstract prototype, the narration is in general extended with certain form of visualization [40]. Alternatives forms of visualization are possible, which will be explained in Section 5.2.

5.2. What sort of media were used to build the information contents into the AP?

The narration, which is one essential component can be presented either textual or verbal or mixed format [39]. The textual information can be presented as static (as a book), as running (as the subtitling in a movie), or animated (appearing and disappearing when needed). The verbal communication can be classified according to having it from a single source (or from one narrator), or from multiple sources (or from a group of actors). Besides this narration, the visual presentation, also called staging of the abstract prototype, plays an important role in the communication. Based on the richness of information, it can be 2D symbol or script-based, 3D model or picture-based, and 4D time-animation (dynamic) based.

The literature reflects the variety of resources that have been used to build abstract prototypes. The visualization can be developed by using paper and other "low-fidelity-materials", or by using any user friendly graphical programming tool. In general we can distinguish two different resources: analog prototypes (e.g. sketches, sticky notes, mock-ups, story boards) or digital prototypes (power point, on-screen animations, life video, motion simulations). Several researchers investigated whether the different ways of visualization and presentation of low fidelity prototypes have had an effect (and what kind of effect) on the outcomes of the usability evaluations [12, 41, 42]. One aspect of this problem is to select the best visualization resources, and another is providing the flexibility with which the media representation of the AP can be changed according to the critics and suggestions of the subjects. The common conclusion of the papers was that the number of usability problems detected is not affected by the kind of media used in the prototyping. Nevertheless, Sellen argued that there are some differences. For example, storyboards require the viewer to fill in details and interpret drawings, while life video leaves less to the imagination. Storyboards allow participants to place themselves in the scene more easily, but with the risk of miscomprehension [43]. We can conclude that the resource

selection should be in harmony with the goal and the manifestation of the abstract prototype, as well as with the programming, computing and sketching skills of the designer, on the industrial application case [44], and on the available computer tools.

5.3. What methods are used for testing AP?

It is a specific characteristic of AP of software that the software is actually evaluated through the abstract prototype. In general it means that the quality of abstract prototyping interplays with the observed quality of the software presented. A poor abstract prototype may imply that the quality of the presented software observed to be less than it actually is. On the other hand, an attractive and perfect-looking prototype can overshadow some quality deficiencies of poorly developed software, because the software itself is not available for demonstration and the designers strive after presenting their concepts as perfect, this paradox situation cannot be avoided. Nevertheless it is important to keep in mind that the quality of the abstract prototype does not have anything to do with the quality of the real system.

The usual first step in testing abstract prototypes is defining the criteria, which in light of the above discussion should be made separate for the demonstrated software and for the demonstrating abstract prototype. As a measure of goodness of the abstract prototype exactness, completeness, fidelity, etc. can be used. The goodness of the developed software however should be evaluated in terms of the operational requirements and usability requirements of the users and the stakeholders [45-47].

The next step of testing the abstract prototype is information gathering based with differently sampled user groups in repeated sessions. For usability testing different methodologies can be used in different contexts; (1) direct experimentation with single or multiple testers at the same time; (2) active information processing, also called creative AP, which counts on the creative contributions of the users in the process; (3) passive information processing, also called demonstrative AP that happens without giving the chance for the participants to intervene or change; and (4) executing the test in a surrounding which is familiar for the testers (in the real world or on the web) or in an unfamiliar lab environment. As discussed in [7, 46], popular methods for information gathering are focus group sessions, field observations , interviews, logging actual use, proactive field study, and questionnaires. The last step involves the evaluation of the test results and making conclusions on the necessary changes. The necessary changes may concern the content of the software and the abstract prototype. What we found in the literature was that the methods used for information processing were in concert with the methods chosen for information gathering.

6 IN WHAT APPLICATIONS WAS ABSTRACT PROTOTYPING USED?

6.1. What are the typical application areas within design support software development?

Numerous design support software (DSS) tools, variously known as CAD, CAE, CAM systems, have been built to support designers in the development process and their number is still growing [48]. Opiyo compared the prototyping of these DSS tools with other software tools. The development of DSS includes requirement analysis and specification, design, implementation, testing, and operation as the main phases of the software development process. At large, the processes of DSS development are analogous to other types of software products. Nevertheless, DSS tools are different as they are based on engineering principles or physical phenomena. It was shown that the design phase of the DSS development process is broad and needs research. Furthermore, as opposed to other software tools, the DSS tools are typically complex and it is difficult to formulate the list of requirements comprehensively at the beginning of development [49]. Design support tool prototyping is actually unique for two reasons. On the one hand, the software engineering paradigms are rapidly evolving and this implies a continuous change in terms of the applications of AP for DSS. As the literature reflects, the original batch-oriented data processing software paradigm became obsolete by now, and gradually gave floor to interactive software applications and, later on, to smartly interacting software tools [11, 40]. As the software paradigms are changing, AP should fulfill different objectives and requirements. On the other hand the method seems to be general enough to use for product-service combinations.

The other aspect is the autonomous or interactive nature of the software. In the case of software tools with programmed (internal) control, logic, the logic of operation is the only important aspect and interaction has hardly any role to play. It means that application of AP for this category of software tools is extremely limited. In the case of interactive software tools, the human interaction happens in the perceptive and motor domain. AP has a role to play here both in the context of designing for interaction and of the interface design. Thus it is necessary to collect the user requirements as well as the users' reflections on the in-process implementation of the tool in order to provide feedback for the software engineers to optimize the concept and the design [15]. Currently the paradigm of intelligent and/or smart software tools has emerged. The interaction with these tools happens not only in the perception and motor domain but also in the cognitive domain. The optimal use of the intelligence embedded in the smart software needs to be achieved and tested through advanced abstract prototyping [32]. As this new paradigm of software proliferates new expectations will emerge against the methodology of AP. These software tools can be supposed to have less interaction with the users but will complement the problem solving activities of the users based on their intelligence [50].

6.2. What are the reported benefits of using AP in these applications?

Several papers have considered the problems of hi-fi prototyping and the benefits of lo-fi prototyping [9, 22]. In comparison with hi-fi prototyping, the low-fidelity abstract prototype can be build fast and cheap. Since a low-fidelity prototype can deal with more incomplete and fuzzy information, it can bring results early in the development process. Challenged by the abstractness in many application cases, the focus of AP is on the presentation of the global concept rather than on the 'fit and finish' issues, or on the details of appearance. It is also a fact that a low-fidelity abstract prototype can be used by many more stakeholders than the high-fidelity prototypes, which are addressing the detailed issues of specific stakeholders. These are important considerations in AP within and between multi-disciplinary teams. AP effectively educates developers to have a concern for usability and formative evaluation. In software development it maximizes the number of times the developers get to refine their design before they commit themselves to coding. Lastly, it is also a benefit that AP avoids that a single bug brings the development to a complete halt. The tests in the early phases of the design also involve that developers still surrender changes on the system.

6.3. What are the reported limitations and pitfalls of AP?

In addition to the immature status of abstract prototyping the literature also reveals certain limitations and constraints which should be taken into account while using AP [17, 51]. Among the most significant ones, we have to mention the following: Firstly, it is not possible to represent all phenomena and effects with low-fidelity prototype, for example to give feedback on whether the user clicked on the right object is hard without having a working prototype. Secondly, low-fidelity abstract prototypes do not require executable code, and do not offer concrete procedures to work with. Moreover as already touched upon in sub-section 3.4, human interaction is needed to convert the abstract prototype into a fully functional prototype or into a real system. Due to this limitation, many people consider that AP is a waste of time and become discouraged. Furthermore AP may seem as unprofessional to some of the end users for the reason that no sophisticated, high technology-oriented approaches are used. Finally it is also a pitfall that abstract prototypes cannot be reused in the detailed phase of software development and the AP methodology cannot be used for testing implementation and operation of the final software.

7 DISCUSSION AND SOME CONCLUDING REMARKS

The objective of this study was to review the state of the art and the various approaches of AP in order to gain insights in the existing definitions, information contents, construction processes, and application opportunities of AP. As explained in Section 3, abstract prototyping (AP), also known as pre-implementation prototyping, low-fidelity prototyping, automatic prototyping, rapid prototyping, early prototyping, low-cost prototyping, surrogate modeling, media prototyping, pre-implementation testing, or paper prototyping is a testing approach in software engineering that supports demonstration and evolution of software concepts at an early stage. It allows designers to optimize the operation of the software and allows end users to understand how to work with the system, and can be manifested as an informal (such as mental or paper models) or as a formal manner (by using language models, animated models or interactive models). AP is used for various purposes in various contexts for

aggregating information, which cannot be obtained otherwise (i.e. without developing prototypes), to attain a comprehensive image on the operation and interaction possibilities, and to formalize the information inquiry and the whole of the software development process. Our belief is that AP is very useful in the development of design support software. We based these beliefs on literature that is shown in Section 6 together with the reported benefits and pitfalls of AP.

Our understanding has been that AP has reached that level of maturity where the need for a comprehensive application methodology can be formulated. The major building blocks have been discussed in Section 4. Below, we make an effort to discuss our findings from the perspective of an application independent AP methodology, and to conclude about the opportunities of combining the relevant knowledge, procedures and methods into a comprehensive methodology for abstract prototyping. Considering the needs of the designers for easy to use support means and for a general applicability, our attention was orientated to a pragmatic methodology. This methodology deals only with the minimally necessary information constructs in the process of abstract prototyping, and can be formulated as:

$$AP=M(N(P,S,C)) \quad (1)$$

Where, AP is the abstract prototype, P are the personas who participate in the process described by the abstract prototype, S is the scenario of all operation and interaction sub-processes taking place in the process, C is the context of the application and use of the software, N is the narration of the story of the contents of the process, and M is media based staging and presentation of the contents of the process. Actually, P, S, and C together constitute the information contents that are needed to describe the operation of and interaction with the developed software. They convey various chunks of information to the abstract prototype, such as: P = (type, sampling, characteristics, attributes), where type $\in$ {end users, knowledge engineers, stakeholders}; S = (system functionalities, user behavior, system-user interactions); and C = (goal of system, tool environment, constraints). In the course of the AP process, first the specific information chunks are collected, structured and interrelated.

Towards the enactment of these contents, these information constructs are converted into and complemented by a narration N, i.e. with a story of the interactions and the autonomous operations happening, and by a media-based representation M, i.e. with an animation and visual presentation of the staging of the happenings, as shown in Section 5. The narration and the visualization work together and strengthen each other. This mixed media representation of the software operation and interaction serves the purpose of demonstrations and assessment. This latter assumes criteria selection, knowledge aggregation from the stakeholders taking part in the early assessment of the software, and processing the feedback for both the software and the abstract prototype. This is important to be mentioned, because the assessment of the software is made through the abstract prototype developed.

Our currents research efforts include the implementation of concrete abstract prototypes and test them with potential users and stakeholders in focus group sessions. In the meantime, the research team published two other papers in which the theoretical framework was presented together with the most important issues of converting theoretical framework into informal structures that can be used as basis for abstract prototyping development [52]. In addition in a second paper it has been shown how abstract prototyping can be used in case of artifact-service combinations [53]. The application methodology was demonstrated and tested through a complex real life example.

REFERENCES

[1] Hakim, J. and Spitzer, T., Effective prototyping for usability. *IEEE professional communication society conference*, pp47-54 (IEEE Educational Activities, Cambridge, Massachusetts, 2000).

[2] Constantine, L., Rapid Abstract Prototyping. *software development*, 1998, 6(10).

[3] Dow, S.P., Heddleston, K. and Klemmer, S.R., the efficacy of prototyping under time constraints. *C&C'09* (ACM, Berkely, California, USA, 2009).

[4] Kyng, M., Making representations work. *Communications of the ACM*, 1995, 38(9), pp46-55.

[5] Constantine, L., What Do Users Want? Engineering Usability into Software. *Windows tech journal*, 1995.

[6] van Notten, P.W.F., Sleegers, A.M. and van Asselt, M.B.A., The future shocks: On discontinuity and scenario development. *Technological Forecasting and Social Change*, 2005, 72(2), pp175-194.

[7] Brandt, E. and Messeter, J., Facilitating collaboration through design games. *Conference on Participatory design: Artful integration*, pp121-131 (ACM, Toronto, Ontario, Canada, 2004).
[8] Irestig, M., Eriksson, H. and Timpka, T., The impact of participation in information system design: a comparison of contextual placements. *Conference on Participatory design: Artful integration*, pp102-111 (ACM, Toronto, Ontario, Canada, 2004).
[9] Rettig, M., Prototyping for tiny fingers. *Commun. ACM*, 1994, 37(4), pp21-27.
[10] Lim, Y.-K., Stolterman, E. and Tenenberg, J., The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas. *ACM Trans. Comput.-Hum. Interact.*, 2008, 15(2), pp1-27.
[11] Opiyo, E.Z., Horvath, I. and Vergeest, J.S.M., Developing software tools for pre-implementation testing of design support tools. 2001.
[12] Sefelin, R., Tscheligi, M. and Giller, V., Paper prototyping - what is it good for?: a comparison of paper- and computer-based low-fidelity prototyping. *CHI '03 extended abstracts on Human factors in computing systems*, pp778-779 (ACM, Ft. Lauderdale, Florida, USA, 2003).
[13] Campos, P.F. and Nunes, N.J., CanonSketch: a user-centered tool for canonical abstract prototyping. In *EHCI-DSVIS 2004.*
[14] Fay, D., Hurwitz, J. and Teare, S., The use of low-fidelity prototypes in user interface design. *Int Symp on Human Factors in Telecommunications*, pp23-31Turin, 1990).
[15] Biddle, R., Noble, J. and Tempero, E., Role-play and use case for requirements review. In *twelfth australasian conference on information systems.*
[16] Buhr, R.J.A., Use case Maps as architectural entities for complex systems. *IEEE transactions on software engineering*, 1998, 24(12), pp1131-1155.
[17] Snyder, C., *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces (Interactive Technologies)*. (The Morgan Kaufmann Publishers, 2003).
[18] Hardgrave, B.C., When to prototype: decision variables used in industry. *Information and Software Technology*, 1995, 37(2), pp113-118.
[19] Constantine, L. and Lockwood, L., Structure and Style in Use Cases for User Interface Design. In Harmelen, M.v., ed. *Object Modeling and User Interface Design.*2000).
[20] Constantine, L., From Abstraction to Realization: Abstract Prototypes Based on Canonical Components -REVISED. *Working paper*, 2000.
[21] Constantine, L., Canonical Abstract Prototypes for Abstract Visual and Interaction Design. *DSV-IS*Madeira, Portugal, 2003).
[22] Haesen, M., Meskens, J., Luyten, K. and Coninx, K., supporting mulitdisciplinary teams and early design stages using storyboards. *IBM developers works*, 2009.
[23] Opiyo, E.Z., Horvath, I. and Vergeest, J.S.M., Knowledge representation and processing in abstract proottyping of design support tools. *ICED 01*Glasgow, 2001).
[24] Weidenhaupt, K., Pohl, K., Jarke, M. and Haumer, P., Scenarios in system development: current practice. *IEEE software*, 1998, March/April, pp34-43.
[25] Constantine, L., Users, Roles, and Personas. In Pruitt and Aldin, eds. *The Persona Lifecycle* (Morgan-Kaufmann, San Francisco, 2005).
[26] Constantine, L., Activity Modeling: Toward a Pragmatic Integration of Activity Theory with Usage-Centered Design. *Interracção 2006*Braga, Portugal, 2006).
[27] Constantine, L.L. and Lockwood, L.A.D., Usage-Centered Engineering for Web Applications. *IEEE software*, 2002, 19(2).
[28] Constantine, L., Conditional Interaction: Improving Use Case Notation for User Interface Design. Application note, 1999).
[29] Li, D., Li, X., Liu, J. and Liu, Z., Validation of requirement models by autmatic prototyping. *Innovations syst softw eng*, 2008, 2008(4), pp241-248.
[30] Diaper, D., Scenarios and task analysis. *Interacting with Computers*, 2002, 14(4), pp379-395.
[31] Sutcliffe, A., Symbiosis and synergy? scenarios, task analysis and reuse of HCI knowledge. *Interacting with Computers*, 2003, 15(2), pp245-263.
[32] Billard, E.A., Patterns of agent scenarios as use case maps. *IEEE transactions on systems, an. and cybernetics-part B: Cybernetics*, 2004, 34(4), pp1933-1939.
[33] Nobrega, L., Nunes, N.J. and Coelho, H., Dialog Sketch: Dynamics of the Canonical prototypes. *TAMODIA'05* Gdansk, Poland, 2005).
[34] Neperud, A., Experience report: Social services application: Essentials and extensions for

improved design. *Constantine & Lockwood, Ltd*, 2002.
[35] Shaw, M., Architectural Issues in Software Reuse: It's Not Just the Functionality, It's the Packaging. (ACM Press, 1995).
[36] Bruneton, E., Coupaye, T. and Stefani, J.B., Recursive and Dynamic Software Composition with Sharing. 2002).
[37] Palanque, P. and Bastide, R., A Design Life-Cycle for the Formal Design of User Interfaces. *Formal aspects of the human computer interface*, 1996.
[38] Dijk, L., Vergeest, J.S.M. and Horvath, I., Testing shape manipulation tools using abstract prototypes. *Design Studies*, 1998, 19, pp187-201.
[39] McCrary, N.E. and Mazur, J.M., Conceptualizing: a narrative simulation to promote dialogic reflection: using a multiple outcome design to engage teacher mentors. *Dducation techn research dev*, 2010, 58, pp325-342.
[40] Haesen, M., Luyten, K. and Coninx, K., Get Your Requirements Straight: Storyboarding Revisited. *Human-Computer Interaction – INTERACT 2009*, pp546-549 (Springer Berlin / Heidelberg, 2009).
[41] Catani, M.B.a.B., D.W. . Usability evaluation and prototype fidelity: Users and usability professionals. In *Human Factors and Ergonomics Society 42nd Annual Meeting, 1998.* .
[42] Wiklund, M., Thurrot, C. and Dumas, J., Does the fidelity of software prototypes affect the perception of usability? In *Human Factors Society 36th Annual Meeting, 1992.* .
[43] Sellen, K.M., Massimi, M.A., Lottridge, D.M., Truong, K.N. and Bittle, S.A., The people-prototype problem: understanding the interaction between prototype format and user group. *Human factors in computing systems conf*, pp635-638 (ACM, Boston, MA, USA, 2009).
[44] Pering, C., Interaction design prototyping of communicator devices: towards meeting the hardware-software challenge. *interactions*, 2002, 9(6), pp36-46.
[45] Robertson, S., Requirements trawling: techniques for discovering requirements. *International Journal of Human-Computer Studies*, 2001, 55(4), pp405-421.
[46] Maguire, M., Methods to support human-centred design. *International Journal of Human-Computer Studies*, 2001, 55(4), pp587-634.
[47] Anastassova, M., Mägard, C. and Burkhardt, J.-M., Prototype evaluation and user-needs analysis in the early design of emerging technologies. *int. conf. on Human-computer interaction: interaction design and usability*, pp383-392 (Springer-Verlag, Beijing, China, 2007).
[48] Horváth, I., Advanced design technologies. In *Mechanical Engineering Conference - Gépészet 2008.* . Budapest, 29-30 May, 2008; G-2008-A1-P-01: p. 1-20.
[49] Opiyo, E.Z., Facilitating the development of design support software by abstrac prototyping. Thesis Industrial design engineering, TU Delft 2003
[50] Lee, E.A., What's ahead for embedded software? *Computer*, 2000, 33(9), pp18-26.
[51] Constantine, L.L., Cutting Corners: Shortcuts in Model-Driven Web Design. In Constantine, L., ed. *Beyond Chaos: The Expert Edge in Managing Software Development* (Addison-Wesley, Boston, 2000).
[52] Horváth, I., Theoretical framework for comprehensive abstract prototyping methodology. *International conference on engineering design, ICED11*Kopenhagen, Denmark, 2011).
[53] Horváth, I., Rusák, Z., Vegte, W.v.d., Opiyo, E.Z. and Koojman, A., Demonstration and assessment of innovation: abstract prototyping of artifact and service combinations. *Journal of Computing And Information Science In Engineering*, 2011.

Contact: Els Du Bois
Artesis university college of Antwerp in collaboration with the Antwerp University
Department of product development
Belgium
M +32 472 56 17 45
T +32 3 205 61 87
els.dubois@artesis.be

Els Du Bois is a PhD student at TU Delft. In 2009 she started her PhD research in collaboration with Delft University of Technology and Antwerp University. Moreover, she coaches Bachelor and Master students in both design and research assignments and is involved in other research projects in design.