

Don't Throw your Software Prototypes Away. Reuse them!

M.J. Escalona¹

University of Seville

Seville, Spain

mjescalona@us.es

L. García-Borgoñon

Ita Innova

Zaragoza, Spain

laurag@itainnova.es

N. Koch

University of Seville

Seville, Spain

nora.koch@us.es

Abstract

The mechanism of prototype development is considered by the research and industrial software communities as a key tool for user-developer communication. In software development, prototypes are used in requirements engineering to help elicit and validate users' needs. Software prototypes like mockups are frequently considered throwaway artefacts and therefore they are often developed very fast, or with very few resources and discarded. In this paper we propose to change this idea, and to create prototypes that can be reused in any model-driven engineering (MDE) process. The paper presents an approach for an automatic mechanism for translating prototype models into requirements models and its implementation in a suitable tool case. This way, software developer teams will be able to dedicate resources to improving communication with users using prototypes because the knowledge acquired will be automatically transferred to the requirements phase of the development process.

Keywords: Software prototypes, model-driven engineering, reusing prototypes

1. Introduction

The concept of prototypes is not something new in software engineering. In fact, prototypes are used in many disciplines as a way to offer a fast preview of the final product [6]. In fields like software engineering, where the product is not a physical product, the idea mainly takes the form of a set of screens or mockups, which more or less accurately represent the structure of the software's future interaction model. There are numerous strategies for developing software prototypes: vertical, horizontal or diagonal, high or low fidelity, evolutionary, fast prototypes, etc. Disciplines like Human-Machine Interaction (HMI) study the advantages and disadvantages of each strategy to try to achieve the best results [4],[8],[11]. Prototypes are intended to make the end user (or the customer) understand what the final software product will be like. Prototyping is a good way to improve communication between software developers and the so-called functional team (customers and end users), but prototypes are frequently conceived of as throwaway artefacts [5]. With exception of evolutionary prototyping the utility of those prototypes once the work with the user has been done is always a subject of debate. The fact that prototypes are destined from the start to end up on the scrap heap means that they are often developed very fast, or with very few resources, and the results of their application are consequently not as good as they could be. However, if we skimp on resources in the process of defining, implementing and validating prototypes, we are reducing the quality of the results they can offer to the outcome of the process. This produces a paradox, investing in good prototypes produces good results, but very often we have to cut back on the resources dedicated to prototypes because they are not destined to be one of the

¹ All authors contributed to the paper. They are ordered alphabetically.

system's final products.

This paper presents a first proposal to try to solve, or at least reduce, this paradox. Our **research question** is “*Could we try to offer an approach to ensure a good ROI (return of investment) in the definition, implementation and validation of software prototypes?*”. To try to answer this question, we propose using the model-driven paradigm to create a mechanism that will ensure that the effort invested in prototype development will be partially recovered in future phases of the software lifecycle development by “reusing” prototypes and the knowledge acquired in their development. We have gone even further as we have defined the metamodels and transformations needed, selected a tool for implementing the prototype, and developed a first version of the transformation engine that generates the requirements model.

Regarding related work García Frey [4] explores in a similar way a model-driven engineering approach for self-explanatory user interfaces using task trees and one-way UsiXML transformations. Another interesting work is the automated comparison of UI prototypes developed with Balsamicq and user stories of Rocha Silva et al. [8] focusing on validation and testing of the user interfaces.

The paper is structured as follows: Section 2 presents a global view of our approach; Section 3 analyses the current research situation and our successes and failures in this area; and finally some conclusions are drawn and future work outlined in Section 4.

2. An Approach for Software Prototype Reuse

This section presents an overview of our approach. In order to offer a suitable mechanism for reusing prototypes, we propose using the model-driven paradigm. The idea of our approach is illustrated in Figure 1.

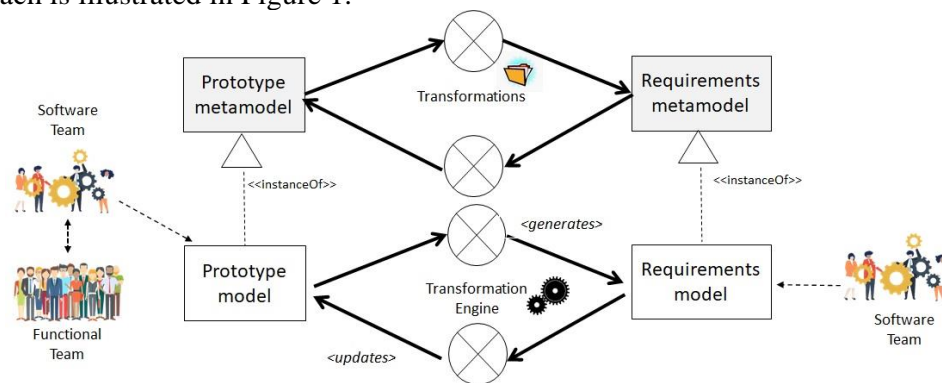


Fig. 1. An overview of our approach

The core idea of this approach is the initial definition of a set of screens or mockups that are the initial software prototype (*Prototype model* in Fig.1). This prototype model has to be defined by the *Software Team* (the team of requirements engineers) in collaboration with the *Functional Team* (the end users and customers who are familiar with the problems to be solved). Ideally, the functional team will interact with the software team, who has to interpret the needs and expectations for the future software product. As a result of this interaction, a prototype model will be developed. The prototype model will be an instance of a *Prototype metamodel* (described in the next section). Although the functional team only sees “screen prototypes”, thus we will actually have structured prototypes, i.e. they concur with the metamodel.

In our approach, we also have another metamodel, the *Requirements metamodel* that represents in an abstract form, the interrelating concepts that are involved in requirements (actors, use cases, objects, actions, etc.). The objective is to generate an instance of this metamodel (shown as *Requirements model* in Figure 1) using artefacts defined in the prototype model. To do this, we propose using a *Transformation Engine*, an engine that will allow us to implement a set of defined transformations using QVT (Query/View/Transformation). With these transformations, we guarantee that the knowledge acquired with the functional team is translated into the requirements model: *prototype knowledge is thus being reused automatically*.

This offers several advantages:

1. We can guarantee that no knowledge is lost in the transition from prototypes to requirements.
2. The transition takes place automatically.
3. A Software Team can dedicate more resources (more time, for instance) to making good prototypes and to really understanding the functional team; more resources to analysing the problem and knowing users' needs and constraints; and more resources to validating the prototype model. They can make the investment because they know that they are going to obtain a suitable ROI. If they have good prototypes they will automatically have good requirements models.
4. Having good requirements models is a critical factor to guarantee the success of a software project. This is widely recognised in the software community [10].
5. Investment in prototype definition also helps to detect early conflicts and to reduce the cost of solving them [7].

Apart from these advantages, there are other, more important aspects to our approach. Figure 1 shows that our transformations engine also considers the possibility of updating the prototype model from the requirements model. This is because we consider it necessary to define bidirectional transformations. In software development, requirements often change or evolve, above all when an agile or an iterative methodology is used. The software team may even find errors or incongruences when they are working on them or translating them into models that are more detailed in the analysis phase. To understand the need for such bidirectional transformations, let us imagine a very typical situation. A functional team develops a prototype model that is translated into a requirements model. The software team finds an incongruence in the requirement model and wants to propose a change. They have two options (if bidirectional transformations are not considered):

1. Make the change in the requirements model, inform the functional team to validate the change and leave the original prototype model as it is. This is the most common procedure in industry, but it has two problems: (1) It produces incongruence between the prototype and the requirements model. (2) Functional teams very often do not understand requirements models very well (they are not usually software experts), so for them it is difficult to understand the changes.
2. Make the change in the requirements model and also in the prototype model, validate the last with the user and regenerate the requirements model. This overcomes the disadvantages described in option 1 but is usually a very expensive process and is therefore not frequently used in industry.

If instead bidirectional transformations are considered, as we do, the software team can make the change in the requirements model, execute the transformation to generate a new version of a prototype model and evaluate it with the functional team. In such a case, the cost is low and consistency between prototype 1 and requirements model is guaranteed.

3. A First Implementation of Our Approach

Reusing software prototypes is not a new idea. It has been studied in several works. As a first step in our research, we performed a literature review that endorsed our research question [9]. In our study, however, we discovered that current works offer no good global solutions.









Firstly, there is little homogeneity in the terminology used. The concepts of software prototypes, mockups, etc, are mixed up. Even the concept of software prototype reuse is not a widely accepted concept, and is frequently used to refer to different ideas. Our idea of prototype reuse is that of a cheap mechanism: that is to say, a mechanism that makes it possible to translate the knowledge obtained through the definition, implementation and validation of software prototypes into the requirement phase as automatically as possible and practically without additional efforts. The knowledge needs to be traceable. If an error or an inconsistency originating in the requirements is detected during analysis, it should be possible to "backtrack" and identify the exact point in the prototype definition or validation where that error or inconsistency was defined and validated. The situation of

terminological heterogeneity makes it very difficult to study the state of the art, so our first failure was to try to find earlier related works or research that might help us. We are currently finishing an SLR (Systematic Literature Review) to extend that initial review.

Secondly, in the state of the art we found very little material relevant to industry. In fact, only toys or academic examples were found in the approaches reported. Therefore, at this point in our research, another **question** arose: “*Is software prototype reuse a good idea for industry?*” Before continuing working with our approach, we tried to answer this question by executing a proof of concept with two companies: an SME (small and medium-sized enterprise) and a big company [10]. Our first success was to realize that industry is interested in this kind of solution, but only if we offer tools supporting the process of reusing prototypes.

With the knowledge obtained from the SLR and the proof of concept, we are now working on our approach. On the metamodel level, we have defined a novel prototype metamodel based on the knowledge acquired. The metamodel was instantiated in a tool called draw.io [1]. We selected this tool after carrying out a comparative study of different tools. draw.io is an open access tool that allows us to define a tool-box where users can use the metamodel, defining classes for its instantiation. For the requirements metamodel we used the NDT (Navigational Development Techniques) [2] requirements metamodel. It is based on WebRE [3], a general requirements metamodel obtained from several approaches. NDT extends WebRE and has been successfully applied it in industry, thus guaranteeing its usefulness. draw.io and NDT-Suite, the tool supporting the NDT methodology offer us the possibility of implementing the transformations needed for our approach. Therefore, we are now working on the implementation of a bidirectional transformation engine: on the one hand, it reads models from draw.io and transforms the concepts into the NDT requirements model; and on the other hand, it reads the NDT requirements models and transforms them updating the draw.io models. We currently have developed a first version of this engine, and are defining the second version following validation with some companies. In Table 1, a simple scenario is presented, which in the first column explains how our approach is used in practice. The second column describes each step, and indicates the tool employed. The last two rows are optional; they are only executed if bidirectional transformations are required.

Table 1. A short scenario for demonstrating our approach

Step	Comment
1. The software team uses draw.io to draw a prototype model using our plugin.	The prototype model is defined according to our prototype metamodel, thus an instance of the prototype metamodel is built internally.  draw.io
2. The software team and functional team work to evaluate the prototype model	This is currently done manually.
3. The software team uses the NDT-Suite to transform the draw.io prototypes into an NDT requirements model.	The transformation engine is executed. It is presented as an Enterprise Architect plugin.  
4. The software team introduces new concepts into a NDT requirements model.	Changes are performed with the Enterprise plugin of NDT.  
5. The software team uses the plugin of NDT-Suite to transform them into draw.io prototypes.	The transformation is carried out in the other direction, from NDT-Suite to draw.io. It is presented as an Enterprise Architect plugin too, but changes are shown in draw.io.   

4. Conclusions and Future Work

Prototyping is a strategy used in a large number of disciplines. For years the software engineering community has acknowledged it as very useful for facilitating communication of functional teams. However, teams' resources for developing prototypes are finite and, in many cases, insufficient. This, coupled with the fact that they, with exception of the case of evolutionary prototypes, are largely considered a throwaway product, means that the information that can be obtained from them is not fully exploited.

In this work we have analysed the potential of prototypes and looked at why this tool must be taken into account in order to understand and meet requirements. This paper presents an early stage approach that proposes a MDE-based mechanism for ensuring that the knowledge from prototypes can be reused. Our approach consists in the use of models and transformations to automatically translate information from prototypes into requirements artefacts. It guarantees that the investment of resources in the definition, implementation and validation of prototypes will be recovered in future phases. With this approach, prototypes are no longer a throwaway item. Having considered bidirectional transformations, we can also offer mechanisms for tracing future changes or future software evolutions, allowing ongoing lifecycle improvements. The paper describes how we are implementing a tool to support our approach. We have validated with industry that this idea can play a relevant role in the software development process [9].

Our emerging idea to reuse prototypes offers many possibilities for future work. We have to finish implementing the approach's architecture and tool. We still need to evaluate the final tool in a real project, and to learn its strengths and weaknesses from academia and industry. Other important future work will be to try to quantify the ROI obtained when using the tool. The starting hypothesis is that investing in prototypes can improve software because prototypes provides a tool for better communication and problem understanding in early stages of the development process. Although this hypothesis is widely accepted, however, we need to find a realistic way to measure such improvements. This is essential for the use of our approach by the industry [10].

Acknowledgments

This research was supported by project AT17_5904_USE, "SocietySoft: Transfer of tools, policies, and principles for creating quality software for the digital society", of the Andalusian Regional Government's Department of Economy, Knowledge, Business, and Universities (Spain) and by the NICO project (PID2019-105455GB-C31) of the Spanish Government's Ministry of Science, Innovation and University.

References

1. DrawIO. <https://drawio-app.com/>
2. Escalona, M. J., & Aragón, G. (2008). NDT. A model-driven approach for web requirements. *IEEE Transactions on software engineering*, 34(3), 377-390.
3. Escalona, M. J., & Koch, N. (2007). Metamodeling the requirements of web systems. In *Web Information Systems and Technologies* (pp. 267-280). Springer, Berlin, Heidelberg.
4. García Frey A. (2010). Self-explanatory User Interfaces by Model-driven Engineering . *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (page 344), NY, USA, ACM
5. Hertel, H., & Dittmar, A. (2017). Design support for integrated evolutionary and exploratory prototyping. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (pp. 105-110).
6. Jensen, L. S., Özkil, A. G., & Mortensen, N. H. (2016). Prototypes in engineering design: Definitions and strategies. In *DS 84: Proceedings of the DESIGN 2016 14th International Design Conference* (pp. 821-830).
7. Mall, R. (2018). *Fundamentals of software engineering*. PHI Learning Pvt. Ltd..
8. Rocha Silva, T., Winckler, M., and Traettberger, H. (2020). Ensuring the Consistency

- between User Requirements and Task Models: A Behavior-Based Automated Approach, Proc. of the ACM on Human-Computer Interaction, Vol. 4, Issue EICS, Article (pp 1–32)
9. Sánchez-Villarín, A., Santos-Montaña, A., & Enríquez, J. G. (2019). Automatic Reuse of Prototypes in Software Engineering: A Survey of Available Tools. In WEBIST (pp. 144-150).
 10. Sánchez-Villarín, A., Santos-Montaña, A., Koch, N., & Casas, D. L. (2020). Prototypes as Starting Point in MDE: Proof of Concept. In WEBIST (pp. 365-372).
 11. Suranto, B. (2015, August). Software prototypes: Enhancing the quality of requirements engineering process. In 2015 International Symposium on Technology Management and Emerging Technologies (ISTMET) (pp. 148-153). IEE