Theoretical aspects of prototyping

M N Orobey¹, A Alchinov², and I M Daudov³

¹Irkutsk National Research Technical University, 83, Lermontov Street, Irkutsk, 664074, Russia
²V.A. Trapeznikov Institute of Control Sciences of Russian Academy of Sciences, Moscow, Russia
³Chechen State University, 17а, Dudaev Boulevard, Groznyy, 364015, Russia.

Abstract. The evolution of programming began at a time when most programs were developed by a single developer. As the complexity of the programs increased, the complexity of their development, debugging and the number of participants in the process also increased, which led to the development of automation systems for programming and organization of teamwork, and later to the development of tools and methods for software design (CASE-tools).

1. Introduction
Software development has become an expensive and costly business over time [3, 4]. It carries its risks for owner and customer in the case of commercial development. The most widespread risks are as follows:

1. The internal difficulties of scheduling, which most often are inadequate timing estimations required to achieve the objectives;
2. The increase or change in the requirements on the part of a customer during the development of a software product can cause the most adverse consequences if the process of software product development is incorrectly set;
3. The problem of employee turnover: replacing a programmer who knows the project with a new member of a team leads, at a minimum, to temporary unplanned expenses;
4. The violation of the specification of software requirements. According to statistics, approximately 15% of projects are closed due to violation of specifications;
5. Low productivity of development team. During the discussion of this problem, the so-called Parkinson's law most often emerges, which states that “work fills the time allocated to it”. According to statistics, when the time allocated to the project is 26 months, the probability of meeting the deadline is 4%. With a 75% probability, the project will end in 38 months, it will not be completed in 15% of cases. One of the time-tested ways to reduce this risk is the transition to the development of a flexible methodology in which tasks are divided into short periods of time, causing a deadline feeling and activating the internal resources of developers.

The desire to improve the development process in order to improve the quality of the finished software product, reduce risks and costs for development, testing and maintenance led to the improvement of automation tools for programming and design, the development of new methodologies and the improvement of management and engineering practices. The risks associated with scheduling, employee turnover and team performance are resolved administratively, through
Experience shows that, the risks associated with the increase or change of requirements on the part of a customer and violation of specifications of software requirements by development team are partially solved by the transition to the development of a flexible methodology through regular discussions of the terms, functionality, requirements and the involvement of an intermediary agent engaged in assessment and resolution of emerging contradictions [2]. Many development teams recognize that the most effective way to solve these problems is to use prototypes in the development process.

2. Determination of software product requirements
The determination of requirements is an important process in the life cycle of any software product. The cost of correcting errors made at this stage increases as a product is ready. Therefore, it is considered good practice to use not only the traditional approach to the collection and analysis of information to transform it into the requirements for software product being developed, but the use of prototypes as an additional means of determination, validation and approvement of requirements.

Firstly, prototypes help to develop unambiguously interpreted requirements, the implementation of which has already been tested by the prototype itself, which in turn allows more accurate calculations of the cost and necessary resources to develop the final software product.

Secondly, the prototype plays the role of a tool for dynamic check of the design as a whole or its individual elements. The development of a prototype makes it possible to quickly get feedback from a customer and/or future users about the design of the project as a whole or any part of it, which in turn allows quick fixing of design errors or expanding the functionality with minimal expenses.

Thirdly, the prototype of a tool for checking the interpretation of program requirements by programmers and extracting new requirements, vague or missed in the early stages of collecting requirements. It follows from this that the prototype allows solving problems of misunderstanding at the early stages of the software life cycle among its participants, avoiding incorrectly interpreted or missed information, and also helps to identify the causes of ambiguity in the interpretation of requirements. For example, understanding the dynamic behavior of a user interface is easier with a prototype than with a text description or graphic models. Development teams using prototypes note that the volatility of requirements defined after prototyping becomes extremely low, which leads to a reduction in corrections in code, specifications and documentation.

According to IEEE guidelines for the development of software requirements, prototypes are useful for the following reasons:

1. A customer may prefer to see and evaluate the prototype than to read and evaluate the specification of software requirements. Therefore, the prototype provides quick feedback.

2. The prototype demonstrates unexpected aspects of system behavior. Thus, it not only answers the questions that arise, but also asks new ones. This helps to more fully analyze and correct the specification of software requirements if necessary.

3. The specification of software requirements based on the prototype tends to undergo less changes during development, thereby reducing development time [1].

Prototypes can be conditionally divided into simulation and software. Prototypes in which real functionality is not implemented, which only simulate it, are usually used by designers in specialized programs. Such prototypes differ in the degree of detail. They are essentially a representation of the design.

3. Simulation prototypes
Paper prototypes sketches on paper, which allows capturing the idea of an interface for further discussion and development. Such prototypes are quickly, free and easy to create. They do not have hardware and software limitations, do not require special knowledge from the participants in the process, changes to them can be made during discussion or presentation to a customer. The purpose of paper prototypes is not to specify ideas, but to define concepts. The more concepts are materialized in
sketchy images on paper, the more options there are for discussion, choice and improvement. Many teams use lecture boards or ready-made sketch boards with a set of templates for this purpose. The disadvantage of paper prototypes is the inconvenience of working with them in a geographically distributed team and their static nature.

Wireframes are low-detail design representations that are quickly created before the first graphic materials and program code. They are the design skeleton that shows which elements are present on the page, their location and purpose. As a result of this, wireframe representations are usually made monochrome so that colors and shades do not distract the team from the discussion of the appearance of specific elements or the color scheme and style. During the discussion of wireframe representations, a team should solve the basic problems on the structure and general idea of the interface being developed, and not be distracted by moments that are not significant at this stage. Due to the fact that wireframe representations can demonstrate user interaction with the interface only in statics, many development teams do not use them, preferring interactive layouts. There are many tools for the creation of wireframe views: Balsamiq, OmniGraffle, Wireframesketcher. Balsamiq provides a library of reusable components that make it easy to create wireframe views. OmniGraffle allows creating wireframe views in various ways, as it provides not only simple tools available for dragging and dropping, but also the ability to use own design through the shape recognition functionality. Wireframesketcher provides stencils for Android, iPhone, iPad and Windows Phone from the gallery of templates; it allows creating own templates, widgets and icons.

A mockup is a medium or highly detailed representation of a design. It visualizes content and demonstrates basic functionality not only in statics, but also with simulated interactivity. There are many tools that allow effective development of layouts of different levels of complexity. For example, Avocode, Axure, Marvel, Mockup.io, InVision, Facebook Origami. Each tool has its advantages and disadvantages, most of these tools are paid. The advantage of using professional tools in the opportunities that they provide developers such as: work on one project with a team of several people, drawing tools, ready-made templates and icons, version control, exporting projects in different formats.

Marvel is an online service that allows transforming layouts and sketches for a web or mobile application into dynamic presentation material. InVision provides simple drag-and-drop tools that simplify the planning of UX design, allow testing the prototype in the browser and providing real-time team work. InVision also supports cloud storage such as Dropbox, Google Drive and Slack. Facebook Origami is a tool for prototyping applications, the interface of which resembles visual programming: a prototype developer has the opportunity to manage ready-made design elements by selecting and connecting them together, without the ability to create new ones.

4. Software Prototypes
Software prototypes are rough developments of a part of a software product’s functionality that can be performed on any platform and technology. Large companies that can afford the costs of the development of software prototypes to test the realisation or feasibility of an idea and the ergonomics of a new interface, have separate teams for the development of prototypes. In addition, the companies with a large code base often use ready-made modules and software components from their projects, collecting from them new versions of products or individual interface elements.

In addition, the development of software prototypes allows making the decision on the choice of a new technology for team or framework. For example, Qwintry LLC, when updating the front-end part of its projects, conducted a study of available frameworks, implementing the registration form in three different versions, using Angular, React and Vue.

5. Implementation of prototypes
In practice, the vast majority of teams use paper prototypes at the very beginning of development to collect and fix ideas and requirements. Some teams do not develop prototypes beyond wireframe representations or layouts, the most advanced teams do not just interactive prototypes, but with
sophisticated functionality and use them at all stages of product life cycle.
The first group uses a prototype as a tool for the extraction and improvement of requirements. The options for the use of the prototype in the second case are more extensive. They use it for:

1. commercial offer, since a visual solution to a customer’s task is more attractive than a text description, which increases the company's competitiveness in the market;
2. testing the feasibility of the idea or its relevance in the market, the development of a pilot project.
3. clarifying the requirements for demonstration to a customer. When a customer interacts with an interactive prototype, it is easier to detect potential problems and controversial issues, as well as eliminate the misinterpretation of functional requirements and customer expectations;
4. testing with end users. At the prototype development stage, any problems determined during testing of the use of the software product are easily fixed in the next iteration.
5. fixing the interface in the technical task in the form of screenshots, as well as individual interface elements and their possible states and entire screens;
6. developing a product according to the prototype in case the development is carried out by the method of evolutionary prototyping;
7. testing the finished product by the testing department;
8. a customer to compare the implementation of the finished product with the prototype agreed upon earlier during the transfer of the software product;
9. adjusting the prototype, in the event of new requirements for the adjustment of the software product.

In practice, the most common methods of interactive prototyping are fast and progressive. Prototype development by rapid prototyping is a quick implementation of basic functionality with little effort to analyze the design of a software product or its functionality, identify requirements or get a feedback. With rapid prototyping, the prototype will be abandoned at some point and will not become part of the finished software product. The main advantage of rapid prototyping is speed. A customer receives the prototype of the interface as soon as possible, he can clarify the requirements and evaluate the idea before starting work on the final version of the software product. In addition it is important that rapid prototyping can be performed within any platform and technology. For the prototype of graphical user interface, both specialized programs (Axure RP, Mockplus, Justinmind, etc.) can be used when the development is carried out by designers and sometimes product managers, as well as a full stack of web technologies (HTML, CSS, JavaScript, one of the popular JavaScript frameworks) when the development team is involved.

6. Conclusion

With progressive prototyping, the prototype should ultimately become a finished product, so the development team must immediately determine which technology stack to use. At each development iteration, a partially implemented software product will be ready not having all the necessary functionality, but demonstrating the implemented functionality. This approach allows conducting development when the requirements are finally unknown and will be determined as the software product is created or the project can be operated in a partially implemented form.

Prototyping is popular, firstly, due to the ability to improve the quality of the specification of software requirements and thereby reduce development time, its cost and risks associated with changes in customer requirements or design errors. Secondly, they are popular due to the ability to combine all the important functions of the system in real-life scenarios of work with it and involve a customer and/or users in the process of the creation of software, thereby getting feedback on the convenience of using the product and the need for functionality in the early stages. Thirdly, it is easier to provoke interest of an investor or customer by a prototype. It increases the flexibility and competitiveness of a team in the commercial software development market.

Fourthly, prototypes simplify the development of new, unparalleled software products, which allow testing their concept, market demand or technical feasibility. Fifthly, they are popular due to the
opportunity to start work on a project in a situation where the requirements are unknown, inconsistent or the development team has not chosen the optimal architecture or software tools and technologies.

During the development of a prototype it is necessary to take into account the shortcomings that will especially manifest themselves in case of the use in an inappropriate project or stage of the software development life cycle. The most common problem with rapid prototyping is that a customer and / or users begin to associate the prototype with the finished product and require appropriate functionality and quality from it. It is also dangerous to delay the prototype development time: waste of time on a too complex prototype, which will be canceled, will undercut all the benefits mentioned above. Therefore, it is necessary to correctly manage the development process, limiting the number of iterations and the time allocated for each of them.

Developing the method of progressive prototyping, common mistakes are the delay of the solution to the most difficult tasks and refuse to refactor the code, which ultimately leads to the accumulation of problems and the mismatch of the final software product to customer requirements. Another mistake is the frequent absence of traditional documentation after completion of development caused by the complexity during the modification of software at later stages of the life cycle.

References
[1] Youn R B, Klyuev R V, Bosikov I I and Dzeranov B V 2017 The petroleum potential estimation of the north Caucasus and Kazakhstan territories with the help of the structural geodynamic prerequisites Sustainable Development of Mountain Territories 9(2) 172–178
[2] Safiullin R N, Afanasyev A S, Reznichenko V V 2019 The concept of development of monitoring systems and management of intelligent technical complexes Journal of Mining Institute 237 322-330
[3] Zhukovskiy Y L, Koteleva N I 2019 Electrical equipment maintenance system with elements of augmented reality technology IOP Conference Series: Materials Science and Engineering 643(1) 012024
[4] Zhukovskiy Y, Batueva D, Buldysko A, & Shabalov M 2019 Motivation towards energy saving by means of IoT personal energy manager platform. Journal of Physics: Conference Series 1333(6) 062033