Analysis of software quality problem

Igor Bogomolov, Maryna Didkovska

Institute of Applied System Analysis, National Technical University of Ukraine, Prospect Peremogy, 37, Kiev, 03056, UKRAINE,
E-mail: igor.bogomolov@gmail.com Site: http://mmsa.kpi.ua

This paper is focused on qualitative software characteristics, what determines software quality and methods to ensure and improve it. In the research the problem was investigated from two points of view: software development and project management. Concerning software development, was formulated what properties have qualitative software and compile the best practices to ensure and improve these properties and overall software quality. Concerning project management, current theory was analysed and found another value – development team, which significantly influences quality. Also, dependency between team productivity and number of team members was stated.

Key words – software quality, qualitative software, software properties, software development methods, project management triangle, project management dimond.

I. Introduction

The main task of the last decade of 20th century and the beginning of the 21st century was to improve the quality of computer services, which possibilities depend on the software. Software was being developed for over fifty years and during this period range of tasks that it can solve, their level of difficulty and presentation of the results changed dramatically.

Today software development is viewed from the perspective of Software plus Service, which provides assembling of software with services in a single, personalized and accessible from anywhere tool. Still development of qualitative software is not the norm and there is no common technology through which developers can create robust software with the relevant costs to time. Sources of defects in modern software are extremely diverse and it only complicates the problem. At the same time the scale of the problem increased. If in the past the price of error in low-quality software was limited by single user or small group, now these boundaries significantly expanded.

Therefore, the relevance of qualitative software development is supported primarily by economic factors. As you know, a lot of industry standards on commercial software include the presence of about 6 errors per 1000 lines of code at the mean of 30 such errors. Can be said, the level of errors in the past 20 years is practically unchanged, despite the use of object-oriented principles, automated debugger, better testing facilities and more control of types in modern programming languages such as Java, Ada and others. According to the report of the National Institute of Standards and Technology USA the volume of economic costs due to faulty software in the US reaches a billion dollars a year, which is estimated about 1% of national gross domestic product.

According to Bill Gates, the relevance of trusted computing has the highest priority in the modern information technologies. This trend has received a special name - the so-called "platform for trustworthy computing". Therefore, the concept of reliable software means software ability to perform tasks assigned to it during requests for their execution. Well-known developer of complex software systems James Fox (IBM) even stated: "The correct software will not tolerate failure".

According to another prominent figure in the computer world David Patterson, who embodies the life of a "renewed after failure computer platforms" (recovery oriented computing), a world rush for a computer performance has led only to human dependency on the technologies. Computers' behavior, ranging from simple devices to powerful routers that support the infrastructure of Internet, is unpredictable. "Computers are ubiquitous for today, the modern world increasingly depends on them, but nobody has proved that they deserve such trust." In his world-famous manifesto David Patterson says: "It is time to elect a fundamentally different basis on which future technologies will be built". And further: "We must develop information technologies, on which the world can really count, as well as it relies on other types of technologies, fully trusting them".

As you can see problem of software quality becomes very important. But what software is considered qualitative? There are different definitions and in this article it would be stated that qualitative software is software that satisfy following properties:

- Compliance with specifications
- Fault-tolerance
- Simplicity to add/change functionality
- Consumption of minimum resources with sufficient performance
- Security
- Usability
- Reasonable price

In this paper problem of software quality will be analysed, and methods that can improve quality and review current project management values that impact it will be described.

II. How to ensure and improve software quality?

For these purpose developers have invented a lot of methods. The research is focused on the following ones:

- Object-oriented programming
- Unified modeling language
- Testing
- Refactoring
- Code review
- Design patterns
- Frequent releases

Object-oriented programming (OOP) is a programming paradigm that uses objects as data structures, consisting of fields and methods together with their interactions, to design computer programs. An object is a discrete bundle of functions and procedures, often related to a particular real-world concept. Other pieces of software can access the object only by calling its functions and procedures that have been allowed to be called by outsiders – object interface. Isolating objects in this way makes software easier to manage and keep track of. OOP strongly emphasizing discrete, reusable units of programming logic. The paradigm focuses on data rather than

processes, with programs composed of self-sufficient modules – classes, each instance of which – objects – contains all the information needed to manipulate its own data structure. An object-oriented program may be viewed as a collection of interacting objects. In it each object is capable of receiving messages, processing data, and sending messages to other objects. OOP includes features such as data abstraction, encapsulation, modularity, polymorphism, and inheritance. This paradigm allows to create a structure of program, that makes possible to divide project scope between team members. Thus, architect creates high-level design of a program, different programmers in charge of different program modules and testers are able to test modules independently. It significantly affects quality because of the specialization of team members. Today OOP is the core consept of all software developmet processes because it makes possible to create more qualitative software faster then any other consept.

Unified modeling language (UML) is a standardized, language-independent notation for the visual modeling of real-world objects as a first step in developing an object-oriented program. Its notation is derived from and unifies the notations of three object-oriented design and analysis methodologies: Grady Booch's notations, James Rumbaugh's object-modeling technique and Ivar Jacobson's use case methodology. Also, it combines techniques from data modeling, business modeling, object modeling and component modeling. UML represents two different views of a software model: static and dynamic. It is widely used to visualy describe structure of a program. That allows to test program on design stage, where cost of errors correction is the lowest. It is an elegant way to ensure program quality on the early stages of development.

Software testing is an software investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks at implementation of the software. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs. Software testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. As such, the methodology of the test is governed by the software development methodology. Different software development models will focus the test effort at different points in the development process. Modern development models often employ test driven development and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. Using tests it is possible to get assessment of almost every software quality aspect: from fault-tolerance to resource consumption and security. Nowadays testing is the easiest method to assess functionality (scope) and quality of a program.

Refactoring is the process of changing the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior. The reason to refactor is to improve the design of software, make software easier to understand, help find bugs and help program faster. Before refactoring a solid set of automatic unit tests is needed. The tests should demonstrate that the behavior of the module is correct. The process is then an iterative cycle of making a small program transformation, testing it to ensure correctness, and making another small transformation. If at any point a test fails, you undo your last small change and try to figure out what is wrong. Through many small steps the program moves from where it was to where you want it to be. In modern software development processes this activity is an integral part of the software development cycle. Like OOP defines the structure of a program and UML describes it, refactoring makes it possible to easily change the structure in order to suit development needs. It is much cheaper and faster than redesign all from the beginning. Well refactored program is much easier to understand, thus maintain, find bugs and add new functionality.

Code review is a phase in the software development process in which the authors of code and other team members get together to review code, improving both the overall quality of software and developers' skills. Reviews can be done in various forms such as pair programming, informal walkthroughs, and formal inspections. Finding and correcting errors at this stage is relatively inexpensive and tends to reduce the more expensive process of handling, locating, and fixing bugs during later stages of development. Reviewers read the code line by line to check for flaws or potential flaws, consistency with the overall program design and adherence to coding standards. Code review may be especially productive for identifying security vulnerabilities. Specialized application programs are available that can help with this process. Automated code reviewing facilitates systematic testing of source code for potential trouble such as buffer overflows, race conditions, memory leakage, size violations, and duplicate statements. Code reviews influence software quality in two ways. First is that during code review less qualitative part of program is being refactored by peers. And second – developers' experience exchange. During code review less experienced developers learned from more experienced ones. However this won't immediately raise software quality, but in long-term outlook it definitely will.

Design pattern is a general solution to a design problem that recurs repeatedly in many projects. Software designers adapt the pattern solution to their specific project. Patterns use a formal approach to describing a design problem, its proposed solution, and any other factors that might affect the problem or the solution. A successful pattern should have established itself as leading to a good solution in several previous situations. In object-oriented programming, a pattern can contain the description of certain objects and object classes to be used, along with their attributes and dependencies, and the general approach to how to solve the problem. Often, programmers can use more than one pattern to address a specific problem. A collection of patterns is called a pattern framework. Patterns directly ensure software quality through usage of well-tested solutions. However, patterns, where they are not needed, can complicate the design of a program and increase development time but not wosen the quality.

Frequent releases is a software development practice that emphasizes the importance of early and frequent releases in creating a tight feedback loop between developers and users. This allows the software development to progress faster, enables the user to help define what the software will become, better conforms to the users' requirements and ultimately results in higher quality software. Frequent releases can also improve the security of a software product, since security fixes are quickly pushed to the end user.

III. How does development team impact on software quality?

If you look at software quality from project management point of view and remember the project management triangle (Fig.1), you will see that there are three values that impact on project quality: scope, cost and time of the project. Let us look at how does development team related to software quality.

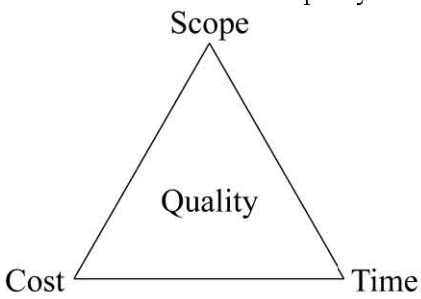

Fig.1 Project management triangle.

You can say, it is "included" in cost or time (with scope no chance). But it is not generally true. To think so, there must be proportional dependency between team and cost or time. With cost it is more less proportional. Developer is paid for his experience, knowledge and skills. But contradiction appears because it is quite hard in the beginning to estimate how experienced and smart developer is. So it is not an exception when more experienced developer "cost" less than less experienced one.

Time from development team dependency is much more complicated. Let us introduce development team productivity variable.

$$Productivity = Work\_done * Time \quad (1)$$

Productivity variable is proportional to the project time variable, so they are interchangeable. And so time from team dependency changes to productivity from number of team members dependency, which is shown on Fig. 2.

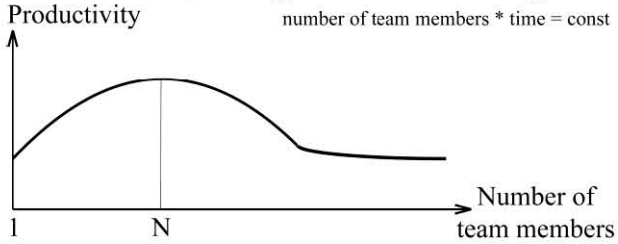

Fig.2 Productivity from number of team members diagram.

On the left part of the diagram (number of team members < N) the productivity increases. It happens because each member has unique specialty and set of skills in the team. And all members are collaborating and helping each other to achieve a common goal.

On the right part of the diagram (number of team members > N) the productivity decreases and then stabilize. It happens because communication between team members increases in arithmetic progression but new members don't bring valuable and unique knowledge anymore. And each member has to spend more time on communication but get no valuable information instead. Stabilization occurs when there are so much people that it is quicker to understand everything by yourself. In this case productivity of the team equals as if each member would work independently.

N is a critical number of members when the last member still brings enough amount of valuable and unique knowledge to the team and communication charges are still acceptable. It heavily depends on scope and complexity of the project.

Based on the previous thoughts it is supposed that there is one more value – development team, which impacts more than others on software quality. Under development team should be understood both number of team members and set of their knowledge and skills. And it is proposed to replace project management triangle (Fig.1) by project management diamond (Fig.3), adding "Team" value.

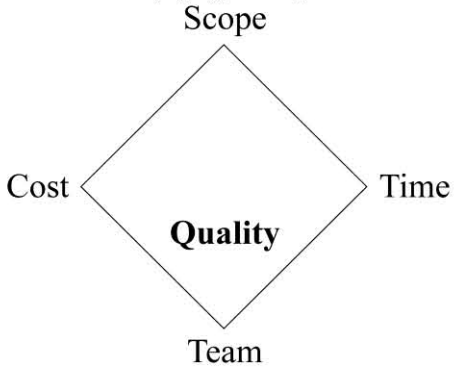

Fig.3 Project management diamond.

IV. Conclusions

1. In this paper have been formulated properties of qualitative software.
2. It has been compiled methods to ensure and improve software quality and described their relation to properties.
3. It has been analysed current values, that influence software quality and dependencies between them and quality.
4. It has been stated new value – development team, that impact software quality as well. And described time to team relation.

V. References

[1] *M. V. Didkovska, Y. A. Timoshenko* "Testing: Basic definitions, axioms and principles. Guidance. PartI"

[2] *M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts* "Refactoring: Improving the Design of Existing Code", Addison-Wesley Professional, 464 pages, July 1999.

[3] *K. Beck, M. Fowler* "Planning Extreme Programming", Addison-Wesley Professional, 160 pages, Oct. 2000.