Usage of CASE approach for guaranteeing of software quality

Pavel Drobintsev, Vsevolod Kotlyarov, Andrey Karpov, Yury Yusupov
Motorola, Saint-Petersburg, Russia
{apd031c, r36959, aak055, ayy302c}@motorola.com

***Abstract**: Today process of a quality guaranteeing is the most complicated in whole software development process. The reasons of such situation are in high requirements for performance and reliability of software, in huge amount of software code which is being developed every day and of course in increasing of software complexity. This paper is devoted to resolve software quality issues by using of CASE tools and the best software development practices within different phases of software development cycle.*
Article is divided in two main parts: the first one presents quality indicators and their relations to software development practices and the second one gives an examples of CASE tools usage in such areas of software development process as requirements gathering and analysis, development and testing. All presented examples based on usage of different CASE tools both internally developed and presented on the market.

1. Introduction

Development of quality software is highly complicated task related to cooperation of huge number of employees which use a lot of processes to perform their tasks. A set of indicators can be used to measure and manage quality. This article is devoted to analysis of these indicators and opportunities to their guaranteeing based on usage of CASE tools and methods.

The main quantitative estimation of software quality is a number of found defects. However in real life estimations and management of quality are based on a set of quality indicators. A lot of them do not have quantitative estimations and are used only based on expert estimations.

2. Quality indicators

Quality indicators related to quality of developed software in terms of developed product properties. Usually the following parameters are mentioned in software development process:

- Efficiency
- Maintainability
- Portability
- Reliability
- Reusability

The other quality indicators such as completeness, understandability, simplicity etc. can be viewed as part of those four. Each of them should be formulated in terms of particular software development project, measured and used for quality estimation. For example simplicity in terms of mobile phone software application is not the same as for PC application, the number of modules and functions in these applications can be strongly different. Let's describe all these qualities related to software products:

Efficiency – is the capability of a system to productively perform its function without wasting system resources. [3] This is one of the significant indicators of software quality in a lot of systems. The highest priority of it is in embedded applications such as automotive applications, mobile phones applications etc. To improve this indicator of the software quality the following software development practices can be used: code generation and requirements verification. Code generation gives an opportunity to use existing optimized algorithms and approaches in developed software. Requirements verification in turn allows to find and check performance critical areas in specifications and move focuse to it during the development process.

Maintainability - is the extent to which software artifacts facilitate change [1]. This particular indicator contains a set of sub sections such as extensibility, simplicity, testability and understandability. The most important for each system is understandability which is the extent to which people can comprehend the behavior of a software system. [2] It is necessary to understand system behavior to make any changes in it. Formalization, reengineering and code generation can be used to achieve this quality indicator. Reengineering approach is used when documentation for maintained system is absent or incomplete. This practice helps to understand system architecture, to improve it, and to fix defects. Formalization jointed with code generation allows generating of enhanced system code based on initial requirements without hard work with legacy code.

Portability – is the extent to which code can operate in different computer configurations and environments [3]. In the modern world software products should work on thousands different hardware platforms and this requires creation of cross platform software. Even if software works in particular local area with limited set of hardware in case the hardware is upgraded in time the software systems should be rebuild in accordance with new requirements. This parameter as long as a maintainability (due to their correlation) can be improved with usage of reengineering, formalization and code generation.

Reliability – is the probability that a program will function without failure over a specified time [1]. In other words it is extent to which developed software workable in normal mode and secure in case of any emergency situations. This particular indicator contains a set of sub sections such as accuracy, robustness and completeness. Completeness – is the extent to which all parts of a software system are fully developed this is the main reason of unspecified works of software. Generally the problems with completeness are in requirements area. Some of them are missed and some are incompleted. The well-known practice to check such problems is verification. The base for verification of requirements is formal specifications which are the prerequisite for formal proving process. Verification of requirements gives a guarantee in completeness of developed software system but in addition it should be checked dynamically by testing.

Reusability – is a measure of the ease with which a software artifact can be used by a software project other than the one for which it was originally intended. [3] It can be guaranteed by creating of well defined interfaces. The main approach for definition of interfaces is based on formal representation in high level graphical or target languages. Reuse of software components is useful if their correctness and quality were guaranteed by previous developers. It can be resolved via implementation of verification and testing automation processes.

3. Software development practices

A software development process contains a lot of practices involved in different areas of software development life cycle. In the article we are focused on modern techniques only which are related to mature software development process. They can be used in all software development projects but require high level of employee's knowledge and existing of special tools.

Formalization – is the process of formal software requirements description. It is usually based on usage of formal graphical languages. The main idea of this practice is in creation of requirements which are formal, understandable to all stakeholders and can be used for future automated development. The most popular graphical formal language for formalization is UML [8]. This standard allows creating system description from different points of view. CASE tools which support UML notation give an opportunity to model system behavior and generate code. Formalization is required for successful start of code generation and requirements verification phases. It is also very useful for testing automation because when formal requirements are created they can be used for tests generation.

Reengineering – is the process of analyzing a subject system to identify the system's components and their relationships and create representation of the system in another form or at a higher level of abstraction. This practice is often used for architecture and design recovering based on legacy code. Existing approaches for code transformation into high level graphical languages allow software developers decrease time for software modernization and support. Technology chain presented in this article shows how CASE tools can be used for reengineering with future specifications and code generation.

Code generation – is the process of software code generation from formal requirements representation. This practice strongly depends on tools because the main problem is in translation of formal requirements into the code. However usage of formal notations for requirements creation is not so simple due to lack of experience in engineers committee.

Testing automation – is the process of test creation, execution, and analysis automation. It is one of the known in presented practices. It has strong tools support. Almost all software development tools contain features for testing automation but testing automation takes a lot of time for customization to target application.

Table 1. Quality indicators and SDP correlation

SDP/Quality	Efficiency	Maintainability	Portability	Reliability	Reusability
Formalization		x	x	x	x
Reengineering		x	x		
Code generation	x	x	x		
Testing automation				x	x
Requirements verification	x			x	x

Requirements verification – is proving of mutual consistency of the requirements. Implementation of this practice into software development process is necessary in case we need to develop system with strong performance and reliability requirements. The main problem here is that engineers are not familiar with the tools which are responsible for verification based on formal notations.. Tools chain presented in this article is based on well known graphical language UML that allows increasing applicability area of this practice.

The table defines correlation between quality indicators and software development practices (SDP).

4. CASE tools in software development

Usage of the CASE (Computer Aided Software Engineering) tools is the most advantage approach in modern software industry. Now these tools already cover such areas as requirements gathering, software testing and verification, code generation etc. They provide functionality for different automation activities and can be used together to improve the software development process. The most number of such tools provide dedicated open interfaces which allow developers to create absent but necessary functionality. We will cover only tools devoted to describe earlier practices. All mentioned tools are being used in Motorola company and have already proved their efficiency:

Klocwork – a tool includes a set of all capabilities for defect finding and architectural analysis. It is designed by Klocwork Company for customers who plan to implement an architecture optimization, a software metrics-based process improvement initiative, or a customized analysis of their software [5]. The tool based on analysis of source code with its intermediate representation in syntax tree. As output user can receive a list of found defects and formal graphical representation of code. Defects finding process is performed using default set of checkers which can be extended by user defined checkers. The benefits of the tool are:

- Possibility to work with huge amount of code. It allows analyzing of systems with millions code lines.
- Existing of open interface for internal data access. The user of tool can access to internal data for adding special features based on data analysis.
- Intermediate syntax of data representation. This very detailed tree received from source code can be used for a lot of reengineering activities.

Klocwork was used within the presented technology chain for defects detecting in code and for reengineering of systems' legacy code.

VRS (Verifier of Requirements Specifications) [10] – is a tool for requirements verification developed in Motorola Company. The tool based on basic protocols theory which is implemented for formal requirements gathering [4]. Basic protocol (BP) is a simple step representation of system behavior. A set of BP can fully describes a system behavior from requirements point of view. Usage of VRS tool allows checking different aspects of BP correctness. From requirements point of view each BP is a part of requirement. From reengineering input of Klocwork each generated BP is a part of system's code which can be analyzed.

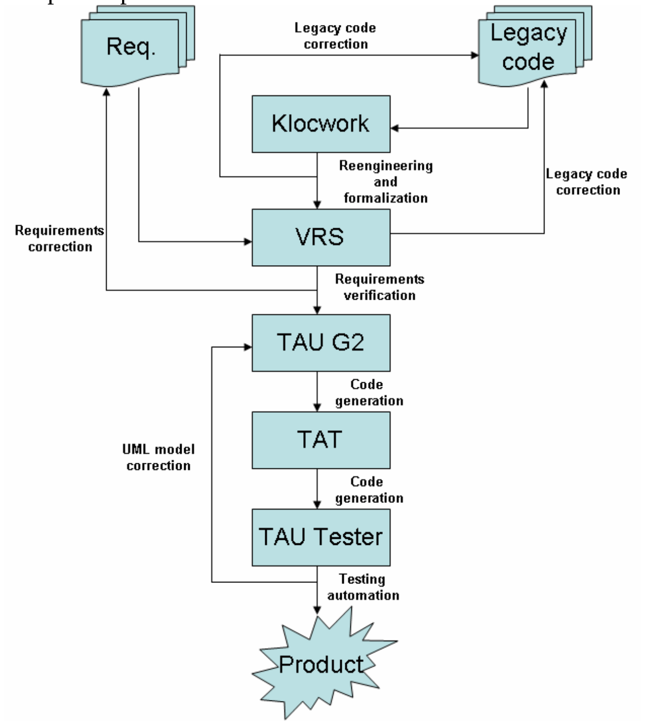
Another way of VRS usage is traces generation from the set of BP. Each trace is a scenario of system's behavior built with BP's concatenation. A trace generated by VRS is presented in the form of MSC (Message Sequence Chart) [9] diagram which is standardized formal language. These traces can be used for future analysis or tests generation.

Telelogic TAU G2 – is a tool for model driven development based on UML (Unified Modeling Language) language. This tool was created by Telelogic Company and initially (in first generation) worked with SDL (Specification and Description Language) [7] specifications. Features presented by TAU G2 allow user to create UML model and to simulate its behavior. After model creation and semantics checking a source code can be automatically generated by TAU G2. Open interfaces allow users to implement additional features into this tool via add-ins. Also the tool can use different compilers to build the generated code. This feature allows generating implementation of one system to different platforms and environments automatically.

TAT (Test Automation Toolset) - is a tool for testing automation based on MSC language usage. TAT was initially developed in Motorola Company for testing automation needs. This tool provides functionalities for automated tests code generation and analysis. Input of TAT is MSC diagrams and output is wrapper and target code of tests. Wrapper is a special code which is responsible for mapping of tests' functions to calls of system under test API. Initially TAT generates skeleton of wrapper which should be customized for target system. The set of tests can be generated on different languages. Multi-language generation based on usage of template mechanism which is a part of toolset. Now generation is available on Java, C, and TTCN (Testing and Test Control Notation) languages.

Telelogic TAU Tester – is a tool for automated testing of software based on TTCN language. This tool was developed by Telelogic Company to support of UML model's testing. Input for the tool is formulated in terms of TTCN notation which is textual representation of tests. Features of TAU Tester allow user to write, execute and analyze tests.

The Pic. 1 shows tools' relations in the software development process.



Pic. 1. Technology chain

There is only one possible configuration of the technology chain which uses all mentioned tools. On the first step legacy code is analyzed by Klocwork. In result code can be corrected in accordance with found defects. Also a special module developed for Klocwork creates a set of BPs generated from source code. This is the reengineering phase of technology which also provides automated formalization. The next phase is verification. At the verification phase BPs created from code or formalized by user from the requirements can be proved by VRS toolset. Defects found in requirements should be corrected in both initial requirements and legacy code. Then they can be used to create a system model and for traces generation for future testing. Code generation phase based on transformation of BP into UML language. That is semi automated task because BP notation is very close to UML state charts. And the final phase is testing automation. Here traces automatically created by verification tool can be used for tests generation in TTCN language and execution with TAU Tester toolset.

In accordance with the process requirements proposed technology chain can be changed and used only for reengineering or testing automation. That is supplied with the features of CASE tools which have very flexible customization mechanism. The main benefit of such approach with usage of different CASE tools is in decreasing software development time and increasing quality of product which is achieved due to automation of different software development practices.

5. Conclusion

Usage of software development practices allows increasing quality of the developed software. These practices are integrated into software development process and require high level of knowledge in engineers committee. Implementation of simple formal notation and supported tools such as UML allow to remove part of complexity limitations.

CASE tools and approaches are available for software development process now. Simplification of tools usage and enhancement of their functionality allow to integrate them into day to day activities of software development process.

The approach presented in the article gives initial knowledge about opportunities to software process enhancement with CASE and formal technologies.

6. References

[1] Weigers, K.E., Creating a Software Engineering Culture, Dorset House, 1996
[2] Pardee, W.J., To Satisfy and Delight Your Customer: How to Manage for Customer Value, Dorset House, 1996
[3] Ronald Kirk Kandt, Software Engineering Quality Practices, Auerbach Publications Taylor & Francis Group 2006
[4] Sergey Baranov, Pavel Drobintsev, Alexander Letichevsky, Vsevolod Kotlyarov, "The verification technology of basic protocols for software design and development", 2005.
[5] www.klocwork.com
[6] www.telelogic.com
[7] ITU-T z.100 (08/2002) // Telecommunication standardization sector of ITU, 202 p. – http://www.itu.int/rec/T-REC-Z.100-200208-I/e
[8] UML Distilled Second Edition. A Brief Guide to the Standard Object Modeling Language.
[9] ITU Recommendation Z.120. Message Sequence Charts (MSC), 11/99
[10] VRS User manual // Motorola, 82 p. - http://www.stc.corp.mot.com/twiki/bin/view/GSGSE/VrsDocumentation