SOFTWARE TESTING AND DOCUMENTING AUTOMATION

Anton Tsybin, Lyudmila Lyadova

Abstract: This article describes some approaches to problem of testing and documenting automation in information systems with graphical user interface. Combination of data mining methods and theory of finite state machines is used for testing automation. Automated creation of software documentation is based on using metadata in documented system. Metadata is built on graph model. Described approaches improve performance and quality of testing and documenting processes.

Keywords: software testing, documenting automation, data mining, dynamic grammar, metadata.

ACM Classification Keywords: D.2 Software Engineering: D.2.2 Design Tools and Techniques; D.2.4 Software - Program Verification; I.2 Artificial Intelligence: I.2.2 Automatic Programming.

Introduction

Lifecycle is the basic concept of software design methodology at the moment. Lifecycle is permanent process that starts from making a decision of creation software system and ends after getting out it from operation.

Lifecycle consists of three groups of processes: main, auxiliary and organizational. Main group includes these steps: creation, set in operation and exploitation, maintenance.

Software creation is also complex step. It includes: lifecycle analysis, software requirements analysis, components structure design, project implementation.

Control testing and acceptance evaluation are realized at the step of information system application and at the adaptable information system exploitation stage. The main goal of this process is software quality assurance.

All described steps of software system creation include auxiliary process of documenting. Documenting process suggests creation of different user instructions and manuals, user and developer documentation.

Software requirements analysis and object model design are the most formalized steps and the most supported by variety of automating tools. Testing and documenting processes are less equipped with automation software. This regularity is peculiar to small software companies. The reason of this peculiarity is complexity of software testing and documenting automation. In spite of plenty of automation software, there is no universal program that suits to any domain. Manual testing and documenting require significant amount of time and funds, but small companies do not have it. However, testing is very important step in software lifecycle, because ignoring it leads to low-quality, malfunctioning software. The Institute of Illinois offered model for measuring of testing automation degree in software lifecycle. This model is called TMM (Test Maturity Model), [1]. It includes a number of methods for improving the testing automation degree.

Documenting process is ignored by many organizations particularly at the final stages of software creation. It harshly decreases efficiency of information system application and maintenance, because it's too hard to change source program code or object model of information system without descriptions of the software functions and requirements when malfunctions are found or system is moved to new hardware architecture or software environment.

Testing and documenting stages become more complicated when user documentation is absent or manuals are made too bad. It leads to decreasing of user leaning efficiency. Therefore variety of data input mistakes and questions to technical support service appear.

Approaches to Software Testing

There is great supply of software intended for automation of programs developing process on the market now. This software automates different stages of program lifecycle. In particular testing of the developing programs at the different stages of a project is important for developers. There are many programs among testing software that verify completeness and consistency for requirements, network interaction and main program functionality

with black-box, white-box and grey-box conformance methods. System reliability under load and overload is testing too. There are a couple of testing programs that also can automate user interface usability testing.

The first method in history of testing was so-called *black-box* testing approach [2]. Black-box testing is suitable for small program modules and is hard to use for complex software systems, because it requires to run many tests to ensure that software system works correct.

New approach for testing in a workmanlike manner was offered by Antony Hoar – *program correctness proof method* [3]. Only one severe weakness of correctness proof was extraordinary complexity of mathematical models construction for big programs. This method was refused after several unsuccessful attempts of applying.

Later some other testing methods based on *revealing errors by source code analysis* were offered. *White-box* conformance testing is well-known method. It was described by Fillis Frankl and Elyne Weyker in 1988 [2]. Common disadvantage of white-box testing is that testing process does not have any information on requirements; therefore there is no information on right results of calculations. So testing software cannot define whenever verified program gave correct results. Thus the results of calculations must be verified manually. The problem is that white-box method uses code coverage and thus it generates plenty of test cases, so it would be impossible to verify them manually. To solve this problem *N-selective test case method* was offered. N-selective test case is based on statistical methods.

Separate group of testing software makes up so-called *record-playback programs*. These programs can record user actions (mouse clicks, key presses etc) and playback them in a user like manner later. For example Rational Robot record-playback program allows checking result correctness with windows controls properties.

Original approaches combining benefits of several testing methods systems exist. These methods make good use record-playback. For example article [4] describes original approach for testing program's graphical user interface. The main disadvantage of this approach is interface state diagrams creation and maintenance complexity.

Approaches for Automatic Documentation Creation

Now the main part of documenting automation software on market is filled up with source code documenting systems. Let's look some of those systems over.

All systems for automatic generation of program documentation use same idea based on grammars. By word "grammar" we will understand any formalized rule set which describes system architecture.

Testing Systems with Fixed Grammar

The majority of modern code documenting systems uses fixed grammars. The idea of fixed grammars is to *extract document information from programming language syntax*. The set of rules that describes programming language syntax is also called *programming language grammar*. These rules are fixed in programming language specifications and thus they cannot be changed.

Let's look at NDoc system as an example. This system is aimed at source code documenting automation for Microsoft dotNET platform. In this example fixed grammar is the set of rules that defines structure of object-oriented program in dotNET and also that describe how to process XML-tagged comments in source code.

The same idea is used by JavaDoc – system for automatic documenting of projects in Java language.

Also there are automatic documenting systems for specific programming languages. LpDoc system generates documentation for List and Prolog languages. Accordingly this system uses grammar that consists of syntax and semantic rules of these languages.

Testing Systems with Dynamic Grammar

Projects documenting automation system based on finite state machines theory is described in [5]. This system allows creating documentation in middle format that can be converted to finite documentation in target format at any time. The idea of saving document data in intermediate format is also used in this research work.

Developer or user creates special templates. Each of templates is the base for documentation. Template includes document structure information. So this documenting system uses grammar made of rules intended for extracting information from file with well-defined structure. The difference from systems based on fixed grammars is that information extracting rules are stored in XML-files outside of documenting system and thus they can be changed at any time forcing the system to change its logic of making documentation.

New Testing and Documenting Automation Methods

This research is given up to testing for GUI programs and automatic generation of user documentation for information systems.

GUI systems were chose for testing automation because most systems now implement graphical user interface. User documentation includes system functions description and step by step instructions of typical operations throw GUI. Thus it is necessary to verify system automatically throw GUI and then generate user documentation on basic functions. Documentation must contain images of user forms from that GUI.

The main problem of testing is to still description of rules which set verifying program's behavior. While small programs for scientific calculations need about 10-20 rules to describe behavior the number of rules describing behavior of complex software systems exceeds hundreds of thousands. In spite of white-box method has some advantages in automatic test cases creation this method can not verify correctness of test execution result automatically. White-box method can do verification of testing results only if a set of rules defining program logic specification exists or somebody can manually check the results.

The concept of researching testing method is based on gathering statistics of some parameters values during program testing. Internal program variables and visual controls properties may be chosen as parameters. Original approach described here is more common than another one which is used by Rational Robot testing system (Rational Robot system does not allow to monitor internal variables values for program under testing).

Source code of program under testing may be used for automatically generating input values for visual interface. For example there is a way to generate several test cases for visual controls using control-flow or data-flow coverage criteria that program's source code will be completely verified. User is required just point out a set of internal variables to verify result correctness.

Input values for interface can be generated with using random values, but it will be less efficient than white-box testing because it isn't aimed at source code bottlenecks.

Testing system collects statistics on selected parameter's values and events of state transitions (like in finite state machines) during program execution. It is necessary to find a way of checking out if each test case returned correct result because we don't have program specification. Data mining method is proposed to solve this problem.

Fuzzy method is selected to identify correctness of test execution because making a program specification is a hard work. This kind of methods is suitable for knowledge extracting from testing results. In other words, we can try to reveal some regularity, associations, dependencies, and sequences in testing results without any manual work. Different conclusions based on exception cases, anomalies in testing results can be made. Also fuzzy methods allow applying criteria for selecting some tests to be verified manually. So testing system will give several tests to be verified by user after making and executing hundreds of test cases throw graphical interface. User returns results back to the system after manual testing and testing system can take this information into account when searching for regularities.

Among data mining technology these methods are suitable for using in testing:

- *Limited selection algorithm*. It checks simple logic events in different data subgroups. Logic events are chosen by heuristic.
- *Associations revealing algorithm*. It recalculates confidence factors for facts combinations based on Bayesian conditional probability and chance.
- *Statistical methods*. They use average values to check distribution statistical parameters, in random distribution hypothesis proving, linear correlation checking.
- *Neural network*. It approximates dependencies by using nonlinear functions.

Program variables dependencies and dependency rules between variables and GUI events can be revealed by applying these methods to program states and state transitions. Given dependencies can be used to find out which variables have values that stand out against revealed rules.

In the context of this research documenting automation component was also created [6]. It allows automatically generate different documents for information systems. At current stage of developing it can generate documents for METAS CASE-system [7].

METAS is software system that has capabilities of design and creation of information systems that can be tuned on different service conditions and user requirements. System METAS uses metadata in interpretation mode to

describe target system data domain, user interface, documentation, business processes. This allows creating and dynamically tuning up information systems with no need in source code changes. Component of automatic documentation generation uses algorithms based on presentation metadata in METAS as multilayer linked lists.

During the research work two-layer documenting component working scheme was offered and implemented. This structure is shown on fig. 1.

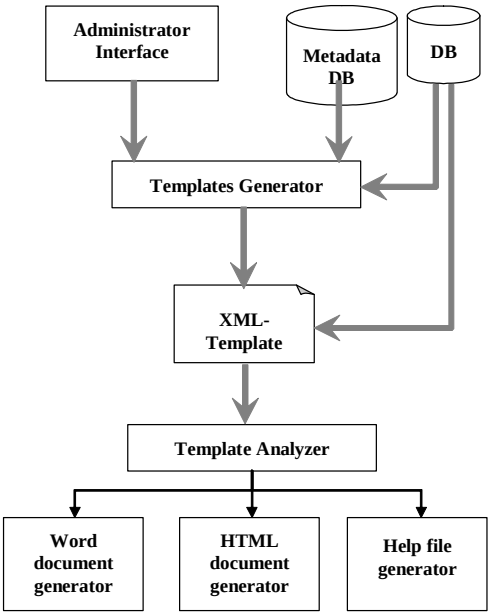

Fig. 1. Common working diagram for component of documentation generation

Implemented working scheme lets the documentation developer setup structure of target document. Document structure parameters are passed as input data into so-called Template Generator. It creates universal view that holds all documentation data from information system. Universal view has format of XML-document. Using XML allows to storage the document in presentation-independent manner. In other words universal view can be converted to user-well-known presentation at any time. For example it can be converted to Microsoft Word document, HTML page or help file.

Documentation developer must have ability to define document structure before generation, namely: ability to construct different hierarchies from documentation elements (chapters), ability to add or delete documentation elements. Special administrator interface was created to allow developer change the document structure.

This interface presents document hierarchy as tree, which can be created using visual interface and textual description by writing tags. Textual description can be synchronized with visual tree.

Administrator interface includes syntax and semantic error checking in document structure (unexpected combination of documentation elements). Semantic errors checking is needed because correct document structure may have limited number of nodes combination.

Administrator interface takes document structure information and transfers of data to XML-template generator. The main task of XML-template generator is creating XML-template, which contains information system description corresponding to document structure given by administrator interface. Template generator goes throw system metadata lists in order written in document structure. Each node in XML-template presents information from one information system object.

Template generator also contains algorithm of searching paths that lead to documenting node in recursive tree. This task is needed because METAS CASE-system has the ability to tune tree of objects (object explorer tree) at main form of created information system. This tree allows users to have easy access to necessary objects of information system. Thus it may be useful to try to describe path to each node in that tree, show the way to find each documenting object in information system's tree.

Template generator can insert images of visual elements (e. g. forms, pages, controls) to generated document by using function of making photo. Visual control image is saved in separate file and then reference to that file is inserted into template.

Next, XML-template analyzer parses existing template file and transfers information system description to end documents generators. Each end document generator transforms given data to target document format (Word, HTML and etc). Document generator must implement fixed interface to be able to receive data from template analyzer. Analyzer uses SAX method of event-based XML- document parsing.

Each node in XML-template describes one object. In addition to node attributes that contain object data each node has special attribute ID which contains index of corresponding object in table of database (fig. 2).

Keeping ID attribute per object allows quickly find original record in data base (DB) during template analysis. This function allows verifying if documented object from template still exists in DB. If object from DB has different data than one from template it may be caused by two situations: table reindexing procedure was executed; manually data changing in template was made. In each situation template analyzer will ask user which of two objects can be taken for documenting.

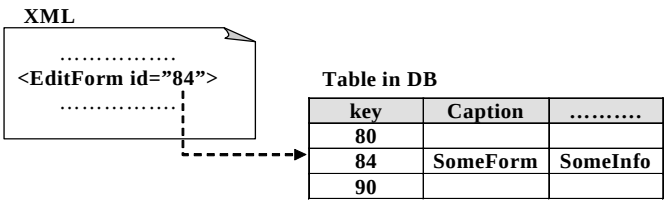

Fig. 2. ID attribute and table key connection

End (target) document generator for Microsoft Word was implemented to demonstrate ability of generating documentation for offered approach. The generator interacts with Word by using OLE. Word document generator takes data strictly from template analyzer.

Implemented documentation component allows:

- to present information system data in universal format (XML);
- to separate processes of extracting information from information system (IS) and documentation generation;
- to convert universal data presentation to any format of document;
- to make documents with different structure;
- to support documentation and database integrity.

Example of offered approach working corresponding to scheme described above is shown on fig. 3.

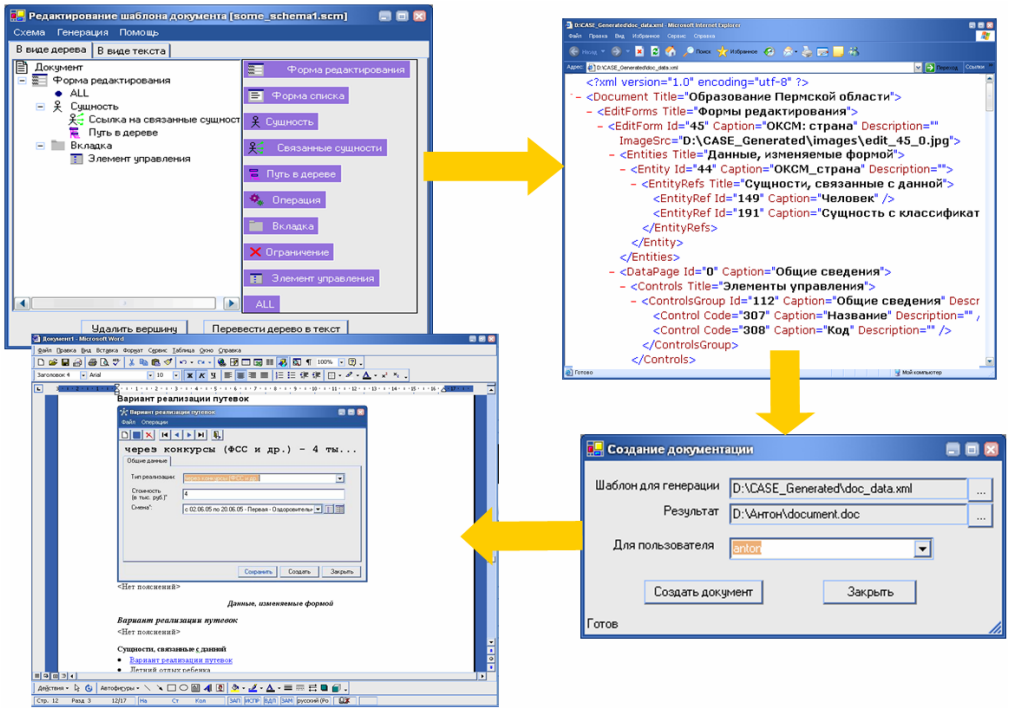

Fig. 3. Example of offered approach working scheme

Process of documentation generation described above consists of these steps:

- document structure description,
- template generation,
- end document generation parameters setting up,
- Word document creation.

Implemented documentation generation component allows creation of different XML-templates containing IS data with required structure. So the component gives to developers the ability of quick generation of documents having different structure and content.

At present time ability of dynamic grammars applying in template generator is researching. Metadata processing algorithm is changed depending on meaning of links between metadata elements. During the research work task of presentation of rules for metadata elements combinations processing was posed. This task requires giving to developer the ability to change this rules. These rules can be presented as information about links between metadata elements (second-level metadata or metametadata). This approach allows implementing documenting component as a separate module which would integrate with any information system based on linked lists. It also offers more opportunities for making documentation structure.

Mathematical model describing dynamic grammars applying in documenting algorithm was developed. Taken decisions were proved formally.

Conclusion

Proposed testing automation method allows test cases generation and execution at any stage of program under testing creation, because of opportunity to choose controlled program variables. Besides regression testing is available if testing system will save previous tests.

Documenting component structure provides further enhancement. During the research work some new ideas about working scheme improvement appeared.

Implemented component can be used for automatic documentation creation in different information systems and also for generating document templates which further can be elaborated to end document by documentation developer.

Bibliography

[1] Дастин Э., Рэшка Д., Пол Д. Автоматизированное тестирование программного обеспечения. М.: ЛОРИ, 2003. С. 15-20.

[2] Gregory M. Kapfhammer. Software testing: Available at: http://cs.allegheny.edu/~gkapfham/research/publish/software_testing_chapter.pdf.

[3] Дейкстра Э. Программирование как дисциплина математической природы: Available at: http://khpi-iip.mipk.kharkiv.edu/library/extent/dijkstra/pp/ewd361.html.

[4] Калинов А.Я., Косачёв А.С. Автоматическая генерация тестов для графического пользовательского интерфейса по UML диаграммам действий: Available at: http://www.citforum.ru/SE/testing/generation_uml.

[5] Суясов Д.И., Шалыто А.А. Автоматическое документирование программных проектов на основе автоматного подхода: Available at: http://is.ifmo.ru.

[6] Цыбин А.В. Автоматическая генерация документации пользователя в информационных системах, управляемых метаданными // Сб. трудов конференции-конкурса «Технологии Microsoft в теории и практике программирования» / Новосибирск: НГУ, 2007. С. 78-80.

[7] Лядова Л.Н., Рыжков С.А. CASE-технология METAS // Математика программных систем: Сб. науч. тр. / Пермь: Перм. ун-т, 2003. С. 4-18.

Authors' Information

Anton Tsybin – Perm State University, Graduate student of the Computer Science Department; Bukirev St., 15, Perm-614990, Russia; e-mail: magicdr@mail.ru.

Lyudmila Lyadova – Institute of Computing, Deputy Director; Podlesnaya St., 19/2-38, Perm-614097, Russia; e-mail: LNLyadova@mail.ru.