ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)
Vol. 2, Issue 3 (July - Sept. 2014)
International Journal of Advanced Research in Computer Science & Technology (IJARCST 2014)

An Approach to Automatic Generation of Test Cases from Use-Cases

[I]Dr. B V Ramana Murthy, [II]Prof. Vuppu Padmakar, [III]A. Vasavi
[I,III]Dept. of CSE, Jyotishmathi College of Technology, and Science, Shamirpet, Hyderabad India
[II]Dept. of CSE, Guru Nanak Institutions Technical, Campus, Hyderabad, India

Abstract
A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal. It consists of a group of elements (for example, classes and interfaces) that can be used together in a way that will have an effect larger than the sum of the separate elements combined. The use case should contain all system activities that have significance to the users. A use case can be thought of as a collection of possible scenarios related to a particular goal, indeed, the use case and goal are sometimes considered to be synonymous

Keywords
Use cases, Gent Case, Tools, Test Case

I. Introduction

A use case (or set of use cases) has these characteristics:

- Organizes functional requirements
- Models the goals of system/actor (user) interactions
- Records paths (called scenarios) from trigger events to goals

Describes one main flow of events (also called a basic course of action), and possibly other ones, called exceptional flows of events (also called alternate courses of action)[3] Is multi-level, so that one use case can use the functionality of another one.

Use case analysis is a technique used to identify the requirements of a system (normally associated with software/process design) and the information used to both define processes used and classes (which are a collection of actors and processes) which will be used both in the use case diagram and the overall use case in the development or redesign of a software system or program. The use case analysis is the foundation upon which the system will be built.

A use case analysis is the primary form for gathering usage requirements for a new software program or task to be completed. The primary goals of a use case analysis are: designing a system from the user's perspective, communicating system behavior in the user's terms, and Specifying all externally visible behaviors. Another set of goals for a use case analysis is to clearly communicate: system requirements, how the system is to be used, the roles the user plays in the system, what the system does in response to the user stimulus, what the user receives from the system, and what value the customer or user will receive from the system. [5]

II. Reasons why use cases are indispensable to your software development project

Use cases help the analysis team, Improve communication among team members. Collaborative effort enhances the success of any team. As the team members work to describe business processes, use cases provide a repository of team members' business knowledge. As a written document, each use case spawns meaningful discussion within the group. The axiom, "the whole is greater than the sum of the parts", applies here. Group discussion exposes in-depth viewpoints that would otherwise remain hidden. With use cases, the team captures these perspectives while identifying the related business goals, conditions, and issues.

III. Encourage common agreement about system requirements

The process of writing and revising use cases produces three important outcomes in the analysis team clarity, consensus, and commitment. Remarkably, it is common for stakeholders to be uncertain about how a process they own actually works! Writing a use case helps stakeholders align the narrative with the details of an existing process.

In a recent project, it became clear that stakeholders could not agree about the specifics of several core processes. However, consensus came quickly as the team wrote and revised use cases.[6] For many stakeholders, these written documents offer a foothold on a sometimes bewildering mountain of complex business processes. Remarkably, use cases often help stakeholders reach common agreement on "best practice" processes as well. In a facilitated group setting, divergent perspectives are welcomed, understood, and appreciated. As a by-product of this agreement, team members inevitably commit to support improved processes to both management and peers.

IV. Reveal process alternatives, process exceptions, undefined terms, and outstanding issues

I always have the analysis team start a use case by developing the "Main Course of Events" (see the sample use case). As the group develops a coherent and ordered set of process steps, team members tend to volunteer statements that begin with the words "what about..." - a clue to previously unidentified "Alternative Paths" to a successful outcome. The "Exception Paths" often arise in a similar way. More of these become obvious when the team considers what happens if any step in the "Main Course" fails. As the facilitator of team meetings, carefully listen for any jargon used by stakeholders. Write these terms down in front of the group and ask for a definition for each one. Later, you'll add these definitions into the project glossary. Also, listen for issues as they arise. Is a process step fuzzy? Is there an area that needs more research, or an item on which team members disagree? Write these down as well and later include them in a project issue log.

V. Expose what belongs outside project scope

Constraints on the project may limit resources and/or the project timeline. As a result, the analysis team may need to prioritize development work, or separate project deliverables into phases.

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

You can help the analysis team by creating a catalog of the use case titles, and arranging them into some meaningful order (e.g., by sub-system or umbrella process). With this catalog, the analysis team can prioritize the use cases. They may decide that some fall outside the project scope, or that some are not needed in the first project phase. Either way, you have given the stakeholders an opportunity to declare which functions they need the most, or which ones they need first.

VI. Transform manual processes into automated processes

When software is being designed to automate aspects of an existing system, the analysis team usually begins by writing "as is" use cases to describe the current business processes. While this effort is time consuming, the result is valuable. Besides revealing the details of an existing business process (including business rules, alternative paths, and exception paths), you will create a launching pad for the team's imagination. As they are writing the use cases, they often discover an improved process, recognize unnecessary steps, or reach agreement on "best operational practices". The "as is" use cases may also allow the system architect to propose high-level process flow diagrams that represent how the new system could work. While the first attempts may not be viable, iterative review and revision by the analysis team may generate a workable architecture for the new system. Use cases help the development team...

VII. Understand business processes

Software developers often lack an understanding of the customer's business. It is easy to forget that software systems should help business people get work done -- effectively, efficiently, and inexpensively. To achieve these objectives, the development team must understand not only the business process the software must support, but also the process' alternatives and exceptions. Use cases provide this information in clear, structured language that developers can readily understand.[2] Use cases also offer a valuable perspective on the stakeholders' business goals, assumptions, and operational rules. These features provide developers with a solid foundation for developing cost-effective solutions to business challenges.

VIII. Recognize patterns and contexts in functional requirements

Developers may view a set of use cases horizontally. For example, one use case requires a customer lookup function. Another use case requires a similar function but with customer data sorted in a different order. The clever development team can find such patterns within a set of use cases. Patterns are often discovered in the "Business Rules" section of use cases as well. Developers may transform these patterns into universal software objects.

As another aid to developers, use cases also reveal operational context. The "Stakeholders Goals", "Pre-Conditions", "Assumptions", and "Post-Conditions" give developers a sense of how the software will be used. By reading these sections, the developer understands what the role identified in the use case is trying to accomplish, and what motivates him or her. Although the analysis team may have prioritized and winnowed the use cases, the development team views them from a far different perspective.[1] As a good development teams writes code, they search for coding efficiencies. While the analysis team may want 12 use cases completed in phase one, the technical manager and the development team may see cost-savings in delivering phase one software for the 12 use cases, plus two more from phase two that are cheaper to build as part of phase one. Of course, the analysis team and the development should jointly consider the effect of this change.

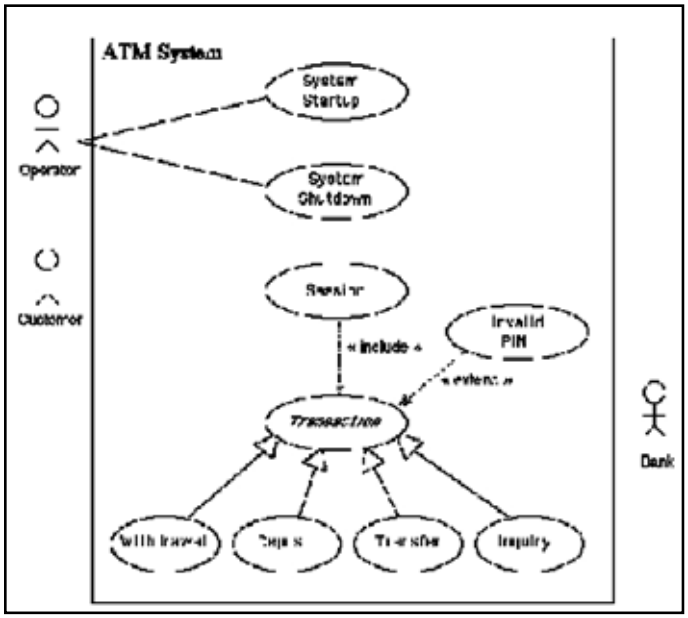
IX. Discover gaps between the requirements and the delivered software

Some years ago, I was asked to join a troubled project in which the system design phase was nearly complete. Unfortunately, detailed functional requirements were nowhere to be seen, and the developers had already begun writing code! In order to catch up, I taught a team of functional users to write use cases themselves. Although we completed the narratives quickly, the developers largely ignored our use cases. That condition changed, however, after the developers installed their first software release. It was clear to us that critical features were missing. The rooky analysis team and I compared the delivered software to our use cases. Although we created a long list of missing features, we challenged the developers to close the gaps rapidly. The next installation was acceptable.

X. Ensure the delivered software works properly

While use cases significantly differ from test cases, the former may be used to derive the latter. The "Assumptions", "Pre-Conditions", and "Post-Condition", "Main Course", "Alternative Paths", "Exception Paths", and "Business Rules" sections are all source material for creating good test scripts. Bundles of use cases organized into system-wide process flows become a source for writing comprehensive endto- end test scripts. As an added bonus, the testing team develops test scripts from the use cases as the development team begins its work. The test scripts are now ready for use as developers complete programs.[8]

Example of use case diagram



Test case:

Test Cases are the implementation of a test case design which will help the software tester to detect defects in the application or the system being tested. This should be the primary goal of any test case or set of test cases. When I write a test case, I think of both

types of test cases, positive test cases and negative test cases. Positive test cases are those which execute the happy path in the application and make sure that the happy path is working fine. Negative test cases as the name suggests are destructive test cases which are documented with some out-ofbox thinking to break the system.[10] A Test Case should be documented in a manner that is useful for the current test cycle and any future test cycles. At a bare minimum each test case should contain: Sr No, Summary or Title, Description, Steps to reproduce, Expected Results, Actual Results and Status of the test case or remarks.

Test Case in Sdlc

Software Development Life Cycle, or Software Development Process, defines the steps/stages/phases in the building of software. There are various kinds of software development models like:

- Waterfall model
- Spiral model
- Iterative and incremental development (like 'Unified Process' and 'Rational Unified Process')
- Agile development (like 'Extreme Programming' and 'Scrum')

Models are evolving with time and the development life cycle can vary significantly from one model to the other. It is beyond the scope of this particular article to discuss each model. However, each model comprises of all or some of the following phases/ activities/tasks.

Test case vs. use cases

A Use Case is not a substitute for a Test Case. I start with this point because there is a growing trend of organizations using Use Cases as Test Cases. Writing Use Cases takes a lot less time, requires fewer resources and less expertise. Use Cases are user scenarios—typical sequences of tasks performed on the software by a typical user. A Use Case is useful for one purpose, in User Acceptance Test (UAT), to verify the software works correctly in typical workflows. Use Cases will include normal flow and alternate flow sequences, but they are still confined to fairly normal end-user workflows. Test Cases cover the software more thoroughly and in more detail than Use Cases. Test Cases include every function that the software is capable of (or is supposed to be capable of); handling every type of data input/output, every expected behavior, every design item, and every class of defect. There are a lot of Requirements that are not covered in Use Cases. But all Requirements must be covered in Test Cases. To satisfy a Test Case, there may be one, two, or more test scripts. Ideally, test scripts have stepby- step, click-by-click instructions that any person off the street could see and instantly perform with no training. (But because of reality constraints, test scripts often assume knowledge common to the designated testers.) When the test scripts pass, the Test Case passes. When the Test Cases pass, the Requirements pass. Every part of a Test Case must be traceable to specific items in the Requirements document, which is not complete until you have captured implied requirements, and converted them into documented Requirements.[9] For example, if I'm testing to verify that closing "print preview" takes the user back to the "print dialog box," then the Requirements document better state that closing "print preview" must take the user back to the "print dialog box." If I am testing the boundary of max characters allowed in a field, the Requirements document better state the max characters allowed in that field. Then you can trace the test back to the requirement.

The tool

The tool, which we call GenTCase (Generator for Test Cases), can be used to layout the usecase diagram of any system. The tool is also able to automatically generate the test cases of the system according to the use-case diagram that has been formed previously. The tool is developed using object-oriented approach with C++ programming language. The tool has 3 major components as shown in The workspace is used as a place for a user to provide the system's requirements by means of a use-case diagram. In the workspace, a Toolbox is used to create, edit and display the use-case diagram. The Toolbox consists of standard symbols and arrows for a use-case diagram such as symbols for an actor and a use case, and arrows for connecting an actor with use cases as well as arrow for generalizations. In the Workspace, a user can also type-in the text for each of the use cases used in the Text Box provided by the tool. The Workspace will allow a user of the tool to layout the use- case diagram according to any system. Once the use-case diagram has been finalized, the user can generate the test cases by using the generator of the tool. The Engine will take all the use cases and search the keywords used in the provided database. The database consists of most standard keywords of a use case. Once the use case used matches the keyword inside the database, the engine will generate its respective test cases according to its use case. Intelligent search technique is used to search all the metadata fields in the entire database.

The intelligent searching technique includes three major processes. First, the keywords are pre-processed by some automatic text operation methods. The result is a collection of metadata, which is considered the logical view of the use case diagram. Next, the metadata describing the logical views are used to construct a metadataoriented index. An index such as this "allows fast searching over large volumes of metadata field".

During the retrieval, the information retrieval engine first performs similar text operations on the user query as those performed on the original use cases. The output of the text operation is a list of metadata, each of which is used to locate, through the index, a list of all the documents in which it occurs. When multiple metadata are present in the query, the search returns the union of the additional information retrieved by all the words. In short, searching is a process of matching keywords in the use cases with those in the query. Lastly, every retrieved metadata is evaluated by its relevance to the query and the additional information of use cases. The way the engine works is by choosing the shortest time-to-locate the object being searched. This will ensure the result returns in few seconds.

The tool will produce the test cases based on the use-case diagram provided in the workspace. These test cases are generated automatically from the tool as the output of the tool. The output is displayed on the screen as well as stored in a file with extension .txt, namely output.txt. A user can open this output file by using a Notepad or Microsoft Word. The output can be used as a checklist for a programmer to test the system that he or she will develop according to the provided test cases. These test cases can also be used to validate the results of the test cases so the requirements of the system are meet.

User who uses the tool can layout the use cases using the Workspace. The Toolbox is used in order to ease the drawing of the use-case diagram. Then, the button for Generator of test cases (GTC) in the Workspace can be used to generate the test cases.[7]

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

XI. Conclusion and Future Work

Gent Case is a tool that is able to generate the test cases automatically according to the system's requirements. The test cases can be used as a checklist for a programmer to validate that the system meets its requirements. The purpose of Gent Case is to reduce the cost of testing the system. However, Gent Case has its limitations where the use cases used are only for functional requirements of a system. The tool is unable to capture the non-functional requirements of a system. Therefore, the nonfunctional requirements need to be captured and tested outside of the tool.

References

[1]. *A. Bahrami Object oriented systems development : using the unified modeling language, Mc-Graw Hill, Singapore. (1999)*
[2]. *C. Nebut, F. Fleurey and Y.L. Traon, Automatic Test Generation: A Use Case Driven Approach, IEEE TRANSACTION ON SOFTWARE ENGINEERING Vol.32, No.3 (2003)*
[3]. *D. Wood and J. Reis (1999). Use Case Derived Test Cases, Software Quality Engineering for Software Testing Analysis and Review (STAREAST99) Online. http://www.stickyminds.com/*
[4]. *I. Jacobson, G. Booch, J. Rumbaugh. The Unified SoftwareDevelopment, England (1992)*
[5]. *J. Gutierez, Escalona M.J. and Torres M.M. An Approach to Generate Test Cases from Use Cases, Proceedings of the 6th International Conference on Web Engineering. pp. 113-114 (2006).*
[6]. *J. Heumann, Generating Test Cases from Use Cases, Rational Software, IBM. (2001).*
[7]. *J. Jansen Using an Intelligent Agent To Enhance Search Engine Performance http://www.firstmonday.org (1996)*
[8]. *R.V. Binder Testing Object-Oriented System. Addison-Wesley. USA (2000)*
[9]. *Rational. (2003). Mastering Requirements Management with Use Cases, Rational Software, IBM.*
[10]. *T. Stanley, Intelligent Searching Agent on the Web, http://ariadne.ac.uk/issue7/searchengine*

Dr. B. V.Ramana Murthy has done his PhD from Osmania University, presently he working as Professor in Computer Science and Engineering, has 18 years of experience in Teaching and R&D. His primary area of interest is Software Engineering & Web Engineering.

Mr. V Padmakar is pursuing PhD in CSE and has done his M Tech (CSE) from JNTUH, presently working as Professor in Computer Science and Engineering has 17 years of experience in Teaching and Industry. His primary area of interests is Software Engineering, Network Security and Data mining

Mrs. A.Vasavi has done her M.Tech (CSE) from JNTUH, presently She is working as Associate Professor in Computer Science and Engineering department, has 10 years of experience in Teaching. Her area of interest is Network Security and Formal Languages.