

# Requirements Use case Tool (RUT)

James R. McCoy  
Software Assurance Technology Center  
NASA Goddard Space Flight Center  
Greenbelt, MD 20771  
301-286-9643

james.mccoy@gssc.nasa.gov

## ABSTRACT

The Requirements Use case Tool (RUT) provides assistance to managers, customers, and developers in assessing the quality of use cases. In addition, RUT serves as a database repository for requirements developed as use cases. To ensure consistency, the tool provides a standard use case template to be used for all use case entry into the repository. Furthermore, RUT provides integration with Rational Rose, the industry-standard tool for developing UML diagrams. The tool also provides a series of metrics useful for calculating information about the relationships among the captured use cases. RUT performs use case evaluation by searching text and identifying risk indicators such as incomplete or weak phrases. The Requirements Use case Tool is a valuable resource for collecting, evaluating, and maintaining software requirements gathered as use cases.

RUT is a web-based, multi-user application that provides project team members with the ability to create, view, and modify use cases and related information for a particular project. The “dashboard” view provided by the tool gives managers and others the ability to quickly ascertain the status of a project by viewing various use case metrics. The tool was developed using multi-platform, open source technologies (PHP and MySQL).

All features of the Requirements Use case Tool described above will be demonstrated at the conference.

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Requirements/Specifications – *methodologies, tools.*

## General Terms

Measurement, Documentation, Verification.

## Keywords

Use cases, requirements analysis, risk indicators.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright is held by the author/owner(s).  
OOPSLA '03, October 26–30, 2003, Anaheim, California, USA.  
ACM 1-58113-751-6/03/0010.

## 1. INTRODUCTION

It is generally considered fundamental by software engineers that requirements are the foundation upon which an entire system is built. It is further presumed that verification and validation are needed to assure that the desired functionality, embodied in the total set of requirements, is ultimately and correctly delivered. All too often, however, development teams fail to satisfy customer requirements, and it is this shortfall, usually discovered quite late in the schedule, that leads to a frantic cycle of fixing, patching, and clock watching. The belief is that correct, complete, and testable software requirements are critical. The success of a project—whether measured in functional or financial terms—can be directly related to the quality of the requirements [4].

The disciplined development and management of software requirements have always been critical in the implementation of software systems—engineers are unable to build what analysts and designers cannot define. It is also generally accepted that the earlier in the life cycle potential risks are identified the easier it is to eliminate or manage the conditions that introduce those risks. Having accepted that requirements of good quality are the basis for successful project development efforts, the objective of work in all phases is to implement those requirements accurately. Studies indicate that problems with requirements are as much as 14 times more costly to fix if they are not found until the testing phase as opposed to finding them in the requirements phase.

The traditional vehicle for capturing and communicating requirements is the Software Requirements Specification (SRS), which is usually a text-based document. The requirements specification, as the first tangible representation of the capability to be produced, establishes the basis for all of the project's subsequent engineering, management, and assurance functions. Whenever the quality of an SRS is poor, it gives rise to risks in all areas of the project.

Today, other formats for capturing and expressing software requirements are gaining popularity. These modern techniques place an emphasis on both the user and the software rather than simply the software's features only. The Unified Modeling Language (UML) [1] provides such a method for capturing requirements through structured text called use cases.

## 2. THE UML AND USE CASES

The UML is the industry-standard, general-purpose notational language for specifying, visualizing, constructing, and documenting the artifacts of complex software systems—

especially large, object-oriented projects. It simplifies the complex process of software design, making a “blueprint” for construction. At the heart of the UML are use cases; the use case diagram is the driver for the eight remaining diagrams that comprise the UML. Use cases form the basis of how the system interacts with “actors” in the outside world, e.g., other users or systems. UML use cases were designed to capture, via a combination of structured text and graphics, the functional requirements of a system. The user-centered analysis provided by use cases is vital to capturing understandable, buildable, and verifiable requirements. Strictly speaking, a use case describes an interaction that provides value to a particular actor [3]. A use case model illustrates a system’s intended functions (use cases), its surroundings (actors), and relationships between use cases and actors (use case diagrams).

### 3. REQUIREMENTS USE CASE TOOL

Extensive research has been conducted in the mechanics of expressing software requirements via natural language and has resulted in various tools—commercial *and* homegrown—for evaluating their quality and usability. That work was extended to apply to use case requirements. This initiative, undertaken by the Software Assurance Technology Center at the NASA Goddard Space Flight Center, was intended to enhance and refine a use case tool that offers a more methodological basis for specifying and managing understandable, buildable, and verifiable functional requirements. The Requirements Use case Tool (RUT) provides assistance to software managers, customers, and developers in assessing the quality of use cases and serves as a project’s database repository for requirements developed as use cases.

RUT was originally developed in 2001 and operated on Microsoft Windows operating systems installed with Microsoft Access 2000. The tool has recently been redeployed as a web-based application, which facilitates a multi-user environment so that various project team members have the ability to create, view, and modify use cases and related information collaboratively for a common project. End users need only have a web browser installed on their computers to access the system—and can do so remotely.

The current tool was developed using multi-platform, open source technologies (specifically, the PHP scripting language and the MySQL relational database management system). This allows the tool to be installed on multiple operating systems and web servers without the need to purchase additional software. Furthermore, this development approach provides users the ability to modify the system to suit their individual project needs and will simplify the process of adding additional features or components to the tool in the future.

To ensure consistency, RUT provides a standard use case template [2] to be used for all use case entry into the repository.

In addition, the tool provides integration with Rational Rose, the industry-standard tool for developing UML diagrams. Individual use cases can be exported from RUT and linked to use case diagram constructs specified in Rational Rose.

One of the most beneficial features of RUT in terms of quality analysis is its capability to parse use case text for matches against a pre-defined and user-modifiable set of risk indicators. Predefined indicators are comprised of words and phrases in the following categories: *options*, which give the developer latitude in satisfying the specification statements that contain them (e.g., *can*, *may*, and *optionally*); *incompletes*, which provide a basis for expanding a requirement or adding future requirements (e.g., *TBD*, *not defined*, and *as a minimum*); and *weak phrases*, which are clauses that are apt to cause uncertainty and leave room for multiple interpretations (e.g., *adequate*, *normal*, and *timely*). The tool also provides a series of metrics useful for calculating information about the relationships among the captured use cases. Included metrics define the levels of *ambiguity*, *completeness*, *volatility*, and *traceability* among use cases.

### 4. SUMMARY

Extensive research has been conducted into writing software requirements of higher quality via natural language. That research has resulted in the development of guidelines for writing more effective requirements as well as tools for evaluating them. However, current software development practices provide a more methodological basis for specifying and managing quality functional requirements. The goal of the Requirements Use case Tool is to provide a clear evaluation technique for software requirements written as use cases. Due in part to its multi-user, multi-platform functionality as well as its extensibility and text-parsing capability, RUT is a valuable resource for collecting, evaluating, and maintaining software requirements gathered as use cases. An effort is currently underway to transition this tool into general usage among NASA’s software project teams.

### 5. REFERENCES

- [1] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.
- [2] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2001.
- [3] D. Kulak and E. Guiney. *Use Cases: Requirements in Context*. ACM Press, 2000.
- [4] W. Wilson, L. Rosenberg, and L. Hyatt. Automated quality analysis of natural language requirements specifications. In *Pacific Northwest Software Quality Conference*, October 1996.