# STORM: Software Tool for the Organization of Requirements Modeling

Sergiu Dascalu, Eric Fritzinger, Narayan Debnath, Olusegun Akinwale

*Abstract*—**Software Engineering is a field of computer science with many areas for expansion and research. One of the most demanding aspects of this field is the specification of software. Attempting to describe and model a software system from scratch can be a time-consuming and tedious process, with many possible causes for problems during the process. STORM, or the Software Tool for the Organization of Requirements Modeling, is a tool designed to streamline the process of specifying a software system by automating processes that help reduce errors. While many programs handle only the diagrams of a system's specification, STORM was designed to maintain the text aspects of the specification, which can be as important as the graphical representations of the system. Details of the tool's requirements and software model are presented in the paper, together with examples of its operational capabilities and snapshots of its user interface. A comparison with related work and pointers to future developments are also included.**

*Index Terms*—**Software Requirements, Specification, Use Cases, Scenarios**

## I. Introduction

Software engineers are faced with design decisions on an almost daily basis. These decisions can have great impact upon a project and so they must be carefully made in order to avoid costly mistakes. To help with their endeavors, the Unified Modeling Language [1, 2] was created as a standard set of notations, diagrams, and text formatting in order for many different developers (and users) to look upon the same diagram and see the same thing. Currently, there are many tools that assist with UML by making diagrams or keeping track of requirements, but text in specification and design tends to be overlooked by these tools.

STORM stands for Software Tool for the Organization of Requirements Modeling. It is intended to be a stand-alone, all-inclusive environment for software engineers that will provide adequate functionality for keeping track of requirements, use cases, and scenarios. The idea is to make STORM as streamlined as possible where all one has to do is go through the tool's interface tabs in order to create the specification of the system with relative ease. On top of keeping track of requirements, it is also intended to maintain use cases and scenarios as well. Other functionality includes keeping a requirements traceability matrix, generating scenarios from use cases, generating the use case diagram based on created actors and use cases, reusing requirements, and (planned for the near future) generating the Software Requirements Specification (SRS) document [3] of the entire software system.

Currently, there are several similar products on the market. Telelogic's DOORS [4] with the Scenario Plus add-on [5] is probably the most representative program in the area of software tools targeted by STORM. DOORS, on its own, maintains requirements and is a very powerful tool for this. With the Scenario Plus add-on to DOORS, use cases can also be maintained, but at a relatively limited capacity. The diagram can also be created using this add-on.

Other UML programs available are diagramming tools, which provide a library of objects to make UML diagrams from. Microsoft Visio [6], Rational Rose [7], and Metamill [8] are just examples of these types of programs. They are powerful software tools for dealing with diagrams, but if a software engineer wishes to rely on more text to describe the system, another program is necessary. STORM is intended to answer this need.

The rest of the paper explains STORM and the various functions provided by the tool. Section II explains the problem that many current programs in the market have in terms of supporting requirements specification and use case modeling. Section III presents the solution that STORM provides via excerpts from the tool's software requirements and specification document as well as through commented snapshots of its user interface. A comparison with similar tools is included in Section IV, while Section V describes our planned future work. Finally, Section VI contains the conclusion of the paper.

## II. The Problem

With all the tools that exist in the market today, only a few of them actually deal with the text aspects of use case modeling. This is a problem because the text in the use cases of a system is very important to the actual modeling. In particular, it supports a better understanding of the system to

Manuscript received February 10, 2006.

Sergiu Dascalu is with the Department of Computer Science and Engineering, University of Nevada, Reno, NV 89557 USA (phone: +1-775-624-0722; fax: +1-775-784-1877; e-mail: dascalus@cse.unr.edu).

Eric Fritzinger is with the Department of Computer Science and Eng., University of Nevada, Reno, NV 89557 USA (e-mail: ericf@unr.nevada.edu).

Narayan Debnath is with the Department of Computer Science at Winona State University, Winona, MN 55987 USA (e-mail: ndebnath@winona.edu).

Olusegun Akinwale is with the Department of Computer Science and Engineering, University of Nevada, Reno, NV 89557 USA (e-mail: walesege@hotmail.com).

be built and helps to bridge the gap between the specification and design phases of software engineering [9. 10]. The flow of events in a use case can even assist with the implementation of the system since it can provide a good indication on how to write part of the code (after all, use cases and scenarios can be seen as higher level pseudo-code descriptions of a system's behavior). If adequate effort is invested in writing the requirements and use case modeling text during the specification phase of a project, the development productivity during the design and implementation phases of the project could be significantly higher.

This is not meant to understate the importance of use case diagrams. These diagrams are very important, as they can visually explain the model of the system and the interaction between the system and its actors (including other systems) [1, 11, 12]. However, without proper text documentation, the diagrams may not have sufficient meaning in the larger scope of things (that is, for interpreting the larger "picture" of the entire system). Thus, it is important to have a clear and well-designed flow of events for each use, not only to explain the purpose of the use case but also to explain how the use case actually performs its job.

Along the same lines, scenarios are important to the modeling of the system as well. Scenarios, which are directly derived from the text of use cases, can show the software engineer problems with his or her design that may not previously have been seen. The primary scenario of a use case can be used almost like an instruction booklet for that particular function of the system. Secondary scenarios can help the designer predict what happens if wrong input is provided to the system or exceptional events occur in the system. Properly designed use cases can generate powerful scenarios to assist with design decisions after the specification phase. This is another reason why the text in use cases is important to the system.

Another missing element of the specification in UML in almost every existing tool in the market is the traceability matrix. A traceability matrix can help to determine just how important a use case is to the entire system and how many requirements depend upon it. With that information, more effort can be directed to a use case that covers several requirements in order to make it more stable and functional. A traceability matrix can also help by showing if there are requirements that are spread over many different use cases, and seeing if perhaps such requirements could be consolidated in order to simplify the system for the user.

The previously mentioned aspects of UML specification (overwhelmingly diagrammatic) are glossed over by almost all current popular software tools. The problem is that those same aspects contain a lot of important information about the system model that needs to be expressed as formatted text. A tool to better handle requirements specification and use case modeling in the text-based component of their description is needed. We offer STORM as a proposed solution.

## III. The Solution

STORM is a stand-alone tool created specifically to handle the text associated with UML models (without discarding the traditional use case diagrams). The tool's primary functionality includes maintaining requirements, actors, and use cases. In addition, the program can keep a traceability matrix, generate scenarios from a use case, and generate the use case diagram from actors and use cases. Given all this functionality, a thorough text-intensive software requirements specification document can be created automatically, which is a significant help for system analysts and designers.

The generating of the use case diagram is one feature that was decided on early. Since little information is needed for the use case diagram, it seemed only natural that a program dedicated to requirements and use cases should be able to do this. There are several properties that are needed in order to generate a diagram, and all these properties can be processed through the user input in STORM.

Another significant feature of STORM is the generation of scenarios. This feature is the primary reason for most of the design decisions made in the program. Generating scenarios can be a powerful function if done properly. With a well-built use case, the engineer could generate scenarios automatically instead of having to rely on a human to copy, paste, and reformat the use case to expand it in several scenarios. There is a lot of chance for human error, which can lead to costly mistakes. Automating this procedure helps avoiding such mistakes.

Requirement reuse has also been added into STORM's functionality via importing and exporting. STORM is capable of sharing requirements with another requirements tool known as RTool [13]. Reuse of requirements is similar to code reuse in that a well-thought out and tested requirement may be reused multiple times in different SRS documents. It can really help in determining what available code can be reused in new software application.

Some excerpts from STORM's own SRS document are included in this section, as well as some commentary on them. Requirements, the use case diagram, and a couple of use cases are included. Several snapshots of STORM's user interface are shown at the end of this section.

### *A. Requirements*

Listed below are some requirements from STORM's requirements and specification document. In our larger SRS document the following priorities were used (not all illustrated in this paper):

(1) Must be completed (already done at this time)
(2) Will be completed in phase II of the project
(3) Additional functionality, to be completed later (phase III)

Functional Requirements (examples):

R01: STORM shall allow the user to add requirements to a list (1)
R04: STORM shall allow the user to add actors to a list (1)

R07: STORM shall allow the user to input individual events in a use case (1)
R08: STORM shall allow the user to include use cases in other use cases (1)
R09: STORM shall allow the user to extend use cases with other use cases (2)
R10: STORM shall allow the user to inherit use cases in other use cases (2)
R15: STORM shall allow the user to maintain a traceability matrix (1)
R16: STORM shall allow the user to generate scenarios from a specified use case (1)
R17: STORM shall allow the user to generate a use case diagram (1)
R18: STORM shall allow the user to import requirements for reuse (1)
R19: STORM shall allow the user to export requirements for reuse (1)

As shown above, STORM was originally designed to handle the text of the specification document. This includes the ability for use cases to include, inherit, and extend one another. The priorities on the requirements were determined by both time available and what is currently completed. Generally, requirements with priority (1) are already completed while requirements with priority (2) are currently being worked on. The requirements with priority (3), not shown in the above list, are part of our future work .

### *B. Use Case Diagram*

The use case diagram for STORM, shown in Fig. 1, contains 18 different use cases. Since there is only one actor on the system, the user, this is a fairly simplified diagram. One thing to note is that three use cases in the diagram have three included (sub) use cases each. These use cases are `Maintain Requirements`, `Maintain Actors`, and `Maintain Use Cases`. Each of them includes the `Add`, `Edit`, and `Delete` (sub) use cases for their individual aspects. By including them in the diagram the writing of the use cases became easier since they are unique in the flow of events that the user must go through to perform them. The `AssociateActor` use case in the diagram requires some additional explanation. STORM allows the user to associate actors with use cases. One can also have actors inherit other actors. Thus, in order to streamline this process, a form was created to allow the user to have actors inherit other actors and associate them with use cases. `AssociateActor` is the use case to explain how to use that form in STORM. Aside from this use case, the other use cases are largely self-explanatory given the requirements listed above.

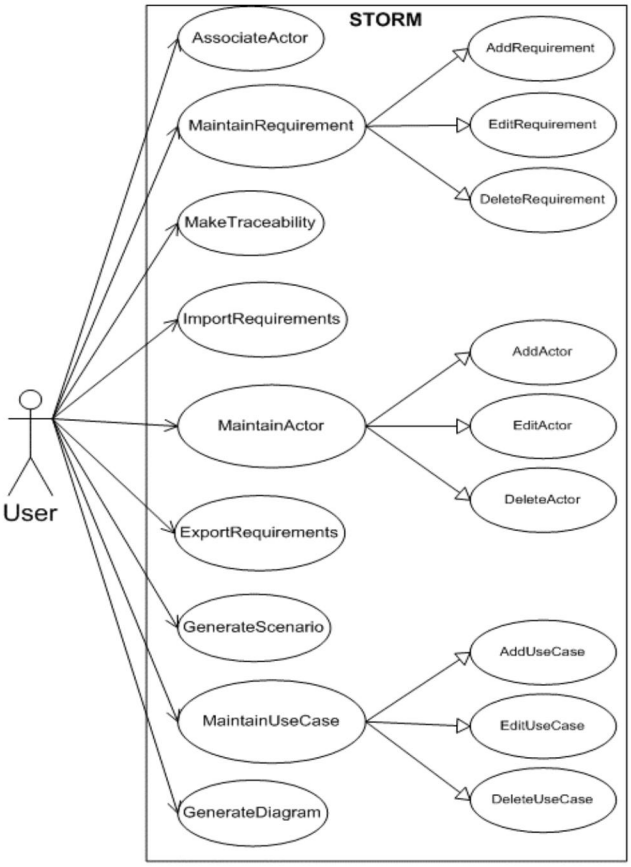


Fig. 1. Use case diagram of STORM

C. *Selected Use Cases*

Two use cases have been selected to illustrate in more depth some of the functionality of the STORM environment. Fig. 2 below shows the `AssociateActor` use case while Fig. 3 presents the `AddUseCase` use case.

| |
|---|
| Name: UC13 – Associate Actor |
| Actors: User |
| Preconditions: User has activated the Associate Actors tab |
| Flow of Events:<br><br>1. User selects an actor from the list<br>2. User selects actors for the current actor to inherit<br>3. User selects use cases that the actor is associated with<br>4. User clicks submit<br>5. System updates information with the actor |

Fig. 2. AssociateActor use case

| |
|---|
| Name:UC10 – Add Use Case |
| Preconditions: User wishes to add a use case |
| Flow of Events:<br><br>1. User inputs the name of the use case<br>2. For each event in the use case<br>a. User selects type of event<br>b. User inputs pertinent fields for the event<br>3. User clicks submit<br>4. If use case name box is empty<br>a. Prompt User to input a use case name<br>Repeat step 1-4 until User inputs a use case name<br>5. System searches for duplicate use case name<br>6. If System finds a duplicate use case name<br>a. Prompt User to input a unique use case name<br>Repeat steps 1-6 until User inputs unique use case name<br>7. System adds use case to list |
| Postconditions: A use case is added to the list |

Fig. 3. AddUseCase use case

D. *Snapshots of the User Interface*

Some of the current operational capabilities of STORM are illustrated next via screenshots of its user interface. Specifically, Fig. 4 presents the opening screen of STORM, which offers access to a number of forms for creating and maintaining requirements, actors, use cases, scenarios, the traceability matrix, and the use case diagram. Fig. 5 provides details on the form used to specify requirements (both functional and non-functional), either from the scratch or by importing (reusing) and updating them.

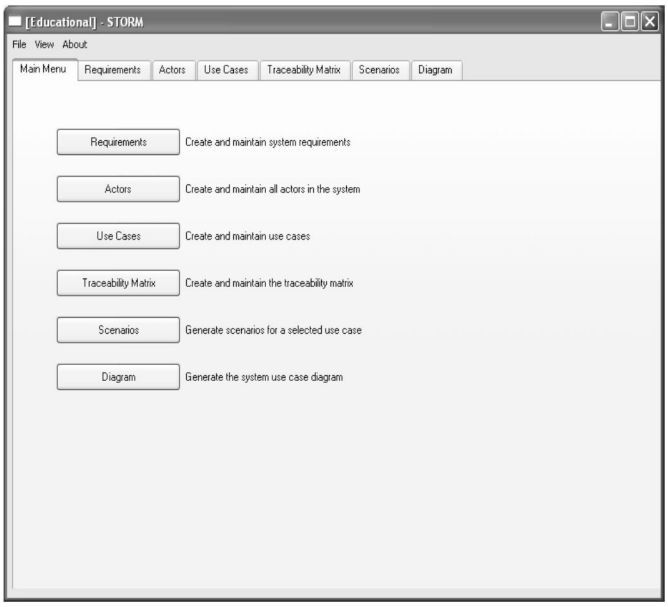


Fig. 4. The opening screen of STORM. From here, the user can easily navigate to any of the forms the program offers.

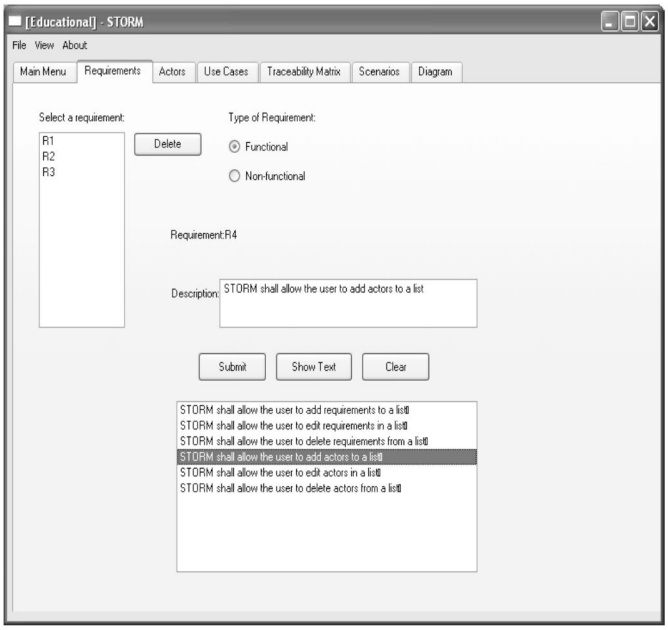


Fig. 5. The requirements form of STORM. Users can import and input requirements, and specify whether the requirement is functional or non-functional.

The use case input form shown in Fig. 6 and Fig. 7 is a key element of STORM. Use cases can potentially be very complicated, depending upon how specific the designer wishes to be. The STORM use case input form caters to this by allowing the user to insert for-loops and if-statements. Also, for operational purposes, the events in these "control blocks" can be "collapsed," as shown in Fig. 7. By simply un-checking the box next to the control block, the user can hide all of the events within the block.

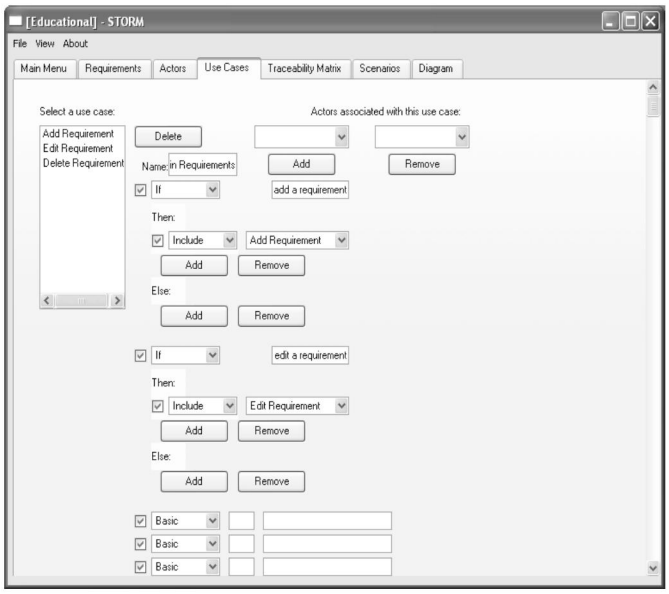


Fig. 6. Use case input form of STORM. Shown here is a use case in progress. The scrollbar on the right assists with navigating longer use cases.

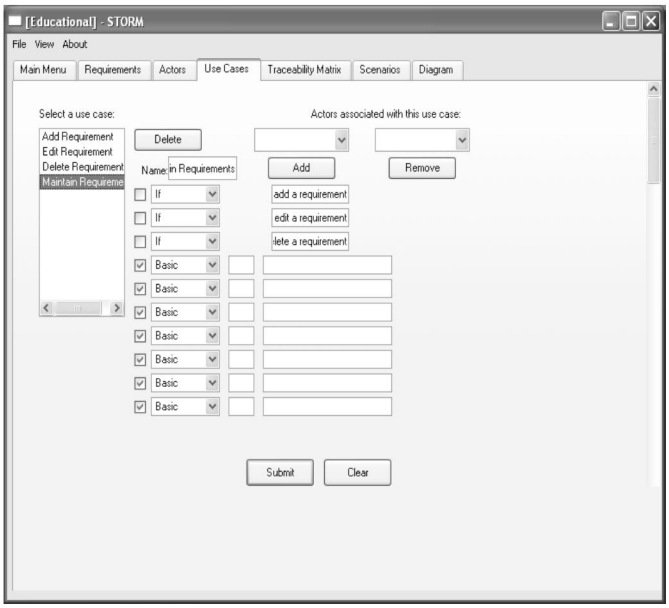


Fig. 7. Use case input form of STORM. The if-statements have been collapsed by unchecking the boxes next to them.

## IV. Comparison With Similar Work

STORM was originally designed to deal primarily with the textual aspects of requirements specification and use case modeling, both major components of the SRS. The only graphical representations that the program is designed for is the use case diagram and the traceability matrix. While STORM may be limited in graphical representations, it makes up for it in the ability to robustly handle the text of the SRS.

RUT, or the Requirements Use case Tool [14], is probably the environment closest in functionality to STORM that we have found. RUT was created by NASA and is able to keep track of requirements, actors, and use cases. It is a web package that can be run on a web server as long as that server has PHP and MySQL. The primary difference is that STORM is also able to generate scenarios from the use cases and the use case diagram from the actors and use cases. Also, the requirements traceability matrix is specific to STORM only. Furthermore, there are many features that will be included in STORM that RUT does not currently support.

DOORS with the Scenario Plus add-on also comes close to STORM in functionality. DOORS is a powerful program, and with Scenario Plus, it can handle use cases as well. However, STORM was specifically designed to handle use cases and scenarios since its inception and thus does not need an add-on. Moreover, STORM allows generation of scenarios from use cases and will soon support the automated generation of the the SRS for the entire system.

## V. Future Work

Further work on enhancing STORM capabilities includes the ability to import from and export to XMI, a standard XML-based file format for diagrams [15, 16]. This is useful if the users want to port their diagrams from one UML tool to another (e.g., from Visio to Metamill). In this way, a carefully crafted use case diagram will not be lost if it is necessary to convert it from one of these tools to STORM or vice-versa.

Misuse cases, which are use cases with malicious intent, are also a direction of future work considered for STORM. Modeling misuse cases is important for applications that depend on security or safety. The idea is to think of potentially hazardous situations and create use cases to mitigate the misuse cases. Given the functionality already available for handling regular use cases, this new feature will most likely not be difficult to implement in STORM.

As previously mentioned in the paper, another important direction of extension will be to allow STORM to generate the SRS document of a software system based on the information already contained in the environment. Essentially, everything that STORM is able to do will be included in the software requirements specification document. This feature will save a significant amount of time in writing, formatting, and revising by generating the document for the users based on the specification work that the users have already performed.

In addition to the above enhancements planned for the near future (summer of 2006), further extensions could include additional "imported" modeling capabilities through the use of plug-ins, a more complex module for handling requirements reuse, and a web-based version of the tool.

## VI. Conclusion

STORM is a robust software tool capable of streamlining the requirements specification phase of the software engineering process. With the ability to import requirements, create and maintain requirements, actors, use cases and scenarios, generate the use case diagram, and generate the primary and secondary scenarios for each use case, STORM is able to save a significant amount of time and allow as little room for error as possible. There are many directions for extending STORM's capabilities, which hopefully will make it one of the most useful tools for performing the complex and demanding software engineering activities of requirements specification and use case modeling.

## References


[1] Object Management Group, "Object Management Group – UML," http://www.uml.org/, accessed December 2005.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language: User Manual,* Addison-Wesley, 1998.

[3] I. Sommerville, *Software Engineering*, 7th ed., Addison-Wesley, 2004.

[4] Telelogic, "Requirements Management & Traceability Solutions," http://www.telelogic.com/corp/products/doors/, accessed Nov. 2005.

[5] Scenario Plus, "Scenario Plus – Templates & Tools for Scenario-Based Requirements Engineering," http://www.scenarioplus.org.uk/, accessed November 2005.

[6] Microsoft Corporation, "Microsoft Office Online: Visio 2003 Home Page," http://www.visio.com/, accessed December 2005.

[7] Rational Software, "IBM Rational Software," http://www.rational.com/, accessed December 2005.

[8] Metamill Software, "Metamill – The UML CASE Tool, Model UML Diagrams and Engineer Java, C, C++, and C# Code," http://www.metamill.com/, accessed December 2005

[9] R. Pressman. *Software Engineering: A Practitioner's Approach*, 4th ed., McGraw-Hill, 2004.

[10] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari, "Application of Linguisitic Techniques for Use Case Analysis," *Proceedings of the 10th IEEE Intl. Conf. on Requirements Engineering*, 2002, pp. 157-164.

[11] J. Arlow and I. Neustadt, *UML and the Unified Process: Practical Object-Oriented Analysis and Design*, Addison-Wesley, 2002.

[12] I. Diaz, O. Pastor, and A. Matteo, "Modeling Interactions Using Role-Driven Patterns," *Proceedings of the 13th IEEE Intl. Conference on Requirements Engineering*, 2005, pp. 209-218.

[13] The Atlantic Systems Guild, Inc, "Currently Available Tools for SRD and System Design", http://www.volere.co.uk, accessed January 2006

[14] Software Assurance Technology Center, "Requirements Use Case Tool (RUT)," http://satc.gsfc.nasa.gov/tools/rut/, accessed December 2005.

[15] O'Reilly Media, "XML.com: XML From the Inside Out," http://www.xml.com/, accessed November 2005.

[16] Structured Technology Group, Inc, "Automated Documentation and Tracking Production System", accessed December 2005. http://www.stgcase.com/casetools/axiomsys.html