Session T4C

An Architecture-Based Software Reliability Modeling Tool and Its Support for Teaching

Wen-Li Wang
Dan Scannell
School of Engineering and Engineering Technology
Penn State Erie, the Behrend College
5091 Station Road, Erie, PA 16563
(wxx18, drr285)@psu.edu

Abstract – This paper presents an architecture-based software reliability modeling tool for pedagogy and demonstrates its support for conveying learning materials to students. Software reliability is an important quality attribute. Improving this attribute early in the software life cycle is highly desirable, because it greatly reduces testing and maintenance effort later on. The architecture-based approach is for such a purpose. This tool enables students to conduct relative analyses on different architectural designs and to compute a more accurate measure once detailed information is available. It equips a GUI for architecture-to-state modeling, taking into account four architectural styles. Students can incorporate additional architectural styles into this framework, and exercise different design alternatives. The GUI shows a graphical representation of software architecture, and helps students visualize the matrix construction for design changes. This tool has shortened students’ learning curve, helped them understand the impact of different designs on reliability estimates, and increased their interests in other quality attributes.

Keywords – Architectural styles, Markov model, Software architecture, Software reliability

INTRODUCTION

Software reliability is a crucial quality attribute and has been a measure for use in quality assurance [4]. Based on IEEE standard definition [8], software reliability is the probability that software performs failure free for a specified time under specified conditions. It is highly desirable to improve software reliability early in the software life cycle to prevent great cost of failures later on. In our software engineering curriculum, software architecture and reliability modeling are two important components. For students to easier learn these topics, a reliability modeling tool that incorporates architectural styles was developed to facilitate students to conduct early software quality prediction. This tool was expected to provide students instant feedbacks and shorten their learning curves.

Many studies have been conducted for software reliability modeling [1, 3, 10, 13]. Some adopt a black-box approach without considering internal system structures, while others exploit a white-box approach to account for software architecture. Many of these models are based on Markov models [2] to compute software reliability. Our tool is a white-box approach following system structures to construct a finite state model. Based upon the state model, reliability is then computed through a number of matrix computations, such as cofactor, transpose, and determinant.

For students, the construction of a state model has been pretty error prone, not to mention the computation errors. The situation gets worse when a variety of interactions and interrelationships among software modules are also taken into account. Many students have suffered from difficulties and frustrations in learning. The long training time with less opportunity for them to practice further delays the conveying of advanced materials. This motivates the initiative of developing this architectural-based software reliability modeling tool. Adhered to our previous studies [12, 13], this tool addresses system structures, helping students compare and comprehend the quality of different architectural designs. Our model is accomplished by extending Cheung’s homogeneous model [1] to further address heterogeneous architecture, including batch-sequential, parallel, fault-tolerance, and client/server styles.

Our tool equips an easy-to-use graphical user interface (GUI) to improve students’ learning, providing a drag-and-drop capability to easily construct a finite state model. Software reliability is then computed accordingly, taking into account the component reliabilities, the transition probabilities among components, and the architectural styles embedded. The GUI eases students to input required argument values and can instantly respond an error message for an invalid input. Such an instant feedback system dramatically improves students’ interests in learning and prevents their pressure from constantly asking questions. The GUI forces students to focus on the important aspects, in order to fill in correct argument values, e.g., students have paid more attention to the discussion of operational profile [9] in order to decide the transition probabilities among components. With the support of the GUI, students learned to conduct relative analyses to evaluate the quality of their alternative architectural designs.

The tool also consists of two vital mathematical libraries, one for matrix computations and the other for reliability computation. The first library can support the matrix computation of determinant, transpose, cofactor, and so on. The second library, along with reliability computation, includes a checklist to help observe the intermediate computation results. Students could use the checklist to decide the desired information to be viewed. The checklist eases the identification of a specific
computation of interest, and assists in illustrating the reliability computation process.

ARCHITECTURE-BASED RELIABILITY MODELING

Software architecture, composed of components, connectors, and configurations, is concerned with high-level design issues [6]. In other words, architecture serves as a high-level abstraction that describes the components and their topologies and interactions. Software Architecture is in our curriculum to introduce students the essence of high-level abstractions. We present software engineers’ design expertise through ten software architectural styles [6, 11] and forty-six design patterns [7]. The styles and patterns help define a family of systems in terms of a framework of structural organization.

In this architecture-based tool, we incorporated four commonly used architectural styles, including batch-sequential, parallel computing, fault tolerance, and client/server. They are easily understood by students, so can increase their interests in others. A batch-sequential style has components executed sequentially in a batch, widely seen in bank systems to perform daily transaction updates. Parallel computing is in a concurrent environment having multiple components running simultaneously to improve performance. A fault tolerance style aims at improving software availability through a set of fault forbearing components compensating for failures of the primary components. A client/server style is similar to a call-and-return procedure, but extends the scope to distributed systems and nowadays web systems.

The following reasons the complexity of computing software reliability for different architectural designs. The complexity hinders students’ interests in learning. Basically, software behaviors are transformed to a finite state model, and software reliability is computed, based on the state model, as the sum of the probabilities of undergoing all transition processes. A state represents the chance of a successful execution of a component or several components during a time period, and a transition advances to the next period. For the reliability computation in this tool, the state model construction conforms to our previous studies [12, 13] to address both homogeneous and heterogeneous software architectures.

The details of reliability computation for individual styles and their integrations are available in [12]. The following only models two components $c_1$ and $c_2$ with successful execution rates $r_1$ and $r_2$, respectively, for the four styles. Figure 1 shows the state models for batch-sequential, parallel, fault tolerance and client/server in order. In Figure 1(a), $c_2$ starts the execution after $c_1$ finishes. Since $c_1$ only transits to $c_2$, the transition probability is equal to 1 and software reliability is computed as $1 \times r_1 \times 1 = r_1 r_2$. In Figure 1(b), inside the dotted oval has $c_1$ and $c_2$ running in parallel. The parallel components are independent so that the reliability is $r_1 r_2$, requiring both components function correctly. Figure 1(c) has components in the dotted rectangle running in fault tolerance. This style requires only one component to be failure free, so that the reliability is computed as $1-(1-r_1)(1-r_2)$ by excluding the failure probability of both simultaneously. For client/server style in Figure 1(d), an execution of client $c_1$ can invoke server $c_2$ from zero to many times. Let the probability from $c_1$ to $c_2$ be $p_{12}$; thus, the other transition probability out of $c_1$ is $1-p_{12}$. It is trivial to see that the probabilities for the two transitions to $c_1$ are both 1. Therefore, the reliability is equal to $(1-r_1(1-p_{12}))/(1-p_{12} r_2)$, derived from $\prod_{n=0}^{\infty} r_1 p_{12} r_2^n (1-p_{12})$.

FIGURE 1: FOUR SOFTWARE ARCHITECTURAL STYLES

Students have been overwhelmed by the time spent on the matrix constructions and computations for large-scale systems. Worst of all, many students were rather reluctant to put effort to carefully analyze different design alternatives. To resolve this situation, students traditionally were trained with application software like MATLAB to facilitate matrix computations. However, their homework results showed that an incorrect reliability measure also came from the inaccurate construction of the transition matrix, not just the erroneous computations. In other words, the actual software behaviors were often represented improperly. Therefore, it is highly desirable to offer students a tool that can support all these needs in matrix construction, matrix computations, and alternative design analyses.

TOOL

The initiative of developing this tool came from teaching software reliability measurement in our Software Engineering and Software Testing courses. The idea of being architecture-based was to support another course Software Architecture.

Session T4C

October 19 – 22, 2005, Indianapolis, IN
This tool shortens students’ learning curve on the state model constructions and reliability computations. The architecture-based modeling helps students understand the tradeoffs among different architectural designs to achieve better design reuse and decision-makings in software development. Our accomplishment is through an easy-to-use GUI design with ability to model different architectural styles. Instant feedbacks are also provided and the display options in a checklist can be turned on/off at runtime to correspond to the teaching needs. This tool has also been a successful story in teaching students GUI designs in the Object-Oriented Design course. It is implemented in Java and the GUI consists of four components: a menu bar, a toolbox, a drawing canvas and an output text area, as shown in Figure 2.

The save feature under File can store the constructed state model and the open can restore it back. This design is based on the snapshot design pattern [5]. Students were also asked to provide an undo feature, based on the command design pattern [5] to back track to a previous state. The Reliability of the Measure option computes reliability and also checks the validity of the constructed state model. A warning message for an invalid state model will be prompted, as shown in Figure 4. This feedback feature saves students time on error checking and guides them to construct a correct model.

### Toolbox Component

The toolbox component consists of a cursor and four drawing objects: link, start, end, and node. They form a group of five buttons, and the selection is mutually exclusive. The cursor is not part of a state model, whose duty is to highlight or drag-and-drop an object in the canvas. A highlighted object can be easily relocated by dragging the mouse, or have its property opened up for change. The functions cut, copy, paste and delete are also performed for a highlighted object.

For the four drawing objects, the start node is the initial state and the end node is the final state of a state model. Their tasks are to facilitate the modeling of multiple inputs and outputs of a software system. Basically, one start is sufficient, so is one end. The start node will link to all the input nodes while the end node will be connected by all the output nodes. The transition probabilities from the output nodes to the end node are 1, and those from the start node to the input nodes will depend on the percentage of going through each input node.

The other two drawing objects are node and link. A node is a regular state that represents one or multiple components running, depending on the architectural style. For each state, the architectural style and reliability of the running component(s) can be specified. Students are allowed to place as many nodes as possible into the canvas. The link object addresses the interrelationships between two nodes by having a connection from one to another. The transition probability of a
link conforms to the Markov property [2]. The value can be either an estimated value or observed from an operational profile [9]. For evaluating alternative designs, students were asked to construct different state models, modify the architectural styles and component reliabilities of the nodes, change the transition probabilities of the links, and compute the reliabilities.

Canvas Component

The drawing canvas is the working area for designing and modifying a state model for software. There can be many nodes and links in it, but with only one start and one end node. The canvas can be resized to accommodate more nodes and links. This working area listens to a number of mouse events and allows drag-and-drop for both nodes and links. The mouse pressed event is to detect a selected object, the mouse clicked event is to open up the property dialog of the selected object, and the mouse dragged event is to relocate a selected object. Students quickly learned how to construct a graphical state model using these three events.

![Figure 5: The property dialog box for a node](image)

Figure 5 shows the property dialog for a regular node. The state editor allows the name of the state to be specified. The radio buttons are for a state to be set to a desired architectural style. The first upper case letters of the styles, ‘B’, ‘P’, ‘S’, and ‘F’, are used in the drawing canvas to indicate the style of each state, as shown in Figure 2. Note that client is not included in the list, because it can be a batch-sequential component, or a set of parallel or fault tolerant components. Even a server can also be a client to ask for services provided by other servers. The dialog enables students to practice different styles, and is a template for them to incorporate more styles and design patterns in their homework practices.

For these four styles, the batch-sequential and the server are considered only one single component running in a state. Therefore, the reliability field can be assigned manually for a component with a value between 0 and 1. The other two styles, parallel and fault tolerance, however will address a set of components cooperating in a state. Therefore, the reliabilities of the individual components are typed in the text area and the value of the reliability will be computed automatically, based on the equation of the chosen style. For example, Figure 5 shows a reliability 0.9702 that is computed as the product of two component reliabilities 0.99 and 0.98 in parallel style. The automatic reliability computation demonstrates students the reliability difference can be affected by the architectural difference, such as between parallel and fault tolerance styles.

![Figure 6: The property dialog box for a link](image)

In addition to a property dialog for nodes, there is property dialog for links to change their transition probabilities. Figure 6 shows the link editor for this purpose. The origin and destination from a state to another state can be specified. The Confirm button checks whether the transition probability conforms to the Markov property. This strengthens the robustness of this tool and relieves students’ error debugging time of the constructed state model. More importantly, students learned how to input a value that is adhered to the Markov property.

Since a link is a directional connection from one state to another, it is likely to have two links with reverse arrow directions between two states. This occurs commonly to a client/server style. In the canvas, a link with a double arrow sign is actually two links with opposite arrow directions. Figure 2 shows one such link, in which the server, denoted with an ‘S’, is called by a client and then returns back the service. For such a link, the property change is by clicking on the associated arrow sign, instead of on the line. A link with one single arrow however can be done in both ways.

Text Area Component

The final component of the GUI is a text area that can display a number of viewing options, as shown in Figure 7. This checklist is activated from the menu bar View and then Options. Following the chosen options, the text area shows only the desired computation results. For the five options, the first one is to show the transition matrix of the constructed state model in the canvas. Based on this transition matrix, the other two matrices \(I - M\) and \((I - M)_{\text{max}}\) can be displayed and the determinants \(|I - M|\) and \(|(I - M)_{\text{max}}|\) can be computed. These two matrices.
determinants are crucial to the aforementioned reliability equation $R = (-1)^{k+1} R_k \frac{|I - M|}{|I - M|}$. 

Table I shows the modeling result of Figure 2 with all options being selected. There are three batch-sequential states, one server, one state for parallel, and one for fault tolerance. All the links have a transition probability 1, except for 0.2 from ‘B’ to ‘S’, 0.5 from ‘B’ to ‘P’, and 0.3 from ‘B’ to ‘F’.

Table I: The Outputs for All Viewing Options

<table>
<thead>
<tr>
<th>The transition matrix $M$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.2 0.2994 0.499 0.0 0.0</td>
</tr>
<tr>
<td>0.0 0.0 0.99 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.9998</td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0 0.9702 0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The matrix $I - M$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.0 1.0 -1.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.0 0.0 1.0 -0.2 -0.2994 -0.499 0.0 0.0</td>
</tr>
<tr>
<td>0.0 0.0 -0.99 1.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0 1.0 -0.9998 0.0</td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0 -0.9702 1.0 0.0 0.0</td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0 0.0 1.0 -0.98</td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0 0.0 0.0 1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The matrix $(I - M)^n$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>1.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.0 1.0 -0.2 -0.2994 -0.499 0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.0 -0.99 1.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.0 0.0 0.0 1.0 0.0 -0.9998 0.0</td>
</tr>
<tr>
<td>0.0 0.0 0.0 -0.9702 1.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0 0.0 1.0 -0.98</td>
</tr>
</tbody>
</table>

The determinant $|I - M| = 0.802$

The determinant $|(I - M)^n| = -0.7677$

In summary, this architecture-based reliability modeling tool provides several advantages to supporting and easing students’ learning.

- **Supporting multiple courses**: This tool can support our three required courses and two tech-elective courses.
- **Instant feedback**: This feature is like having a tutor guiding students at all times to set up a correct model. Students do not feel embarrassed or intimidated, and are confident in self-learning.

SUPPORT FOR TEACHING

The development of this reliability modeling tool has taken into account its potential supports to a number of our course outcomes. Since building a tool is rather time consuming, the ambition was to make the best use out of it. Therefore, the design of the tool was not only for software reliability modeling, but also for the tool to be used in other topics. By far, we have exercised its use in three required courses and one tech-elective course, and are planning to explore the possibility in an upcoming tech-elective course. The three required courses are *Object-Oriented Design*, *Software Engineering*, and *Software Verification, Validation and Test*. The exercised tech-elective course is *Software Architecture* and the one in plan is *Software Quality Assurance*.

In the *Object-Oriented Design* course, this tool was used to demonstrate the GUI design and the advantages of encapsulation, inheritance, and polymorphism. In the *Software Engineering* and *Testing* courses, the tool assisted students in learning finite state machines and software reliability modeling. The tool itself also served as a target system under test with a number of instrumented faults to support the teaching of functional testing and test coverage. In the *Software Architecture* course, it served as a template for students to incorporate more architectural styles and to discuss the heterogeneity of different styles’ combinations. In addition, the high-level architectural designs displayed in the tool also helped convey the knowledge of lower level design patterns.

In summary, this architecture-based reliability modeling tool provides several advantages to supporting and easing students’ learning.
Shorter learning time: The GUI and its drag-and-drop capability allow students to quickly design a state model and understand the construction of a transition matrix.

Architectural design: The architectural styles embedded in the tool help students understand and discuss the characteristics of alternative architectural designs.

Reliability modeling: This tool introduces students a white-box approach to compute software reliability, and raises their interests in other approaches.

Experiments: The tool facilitates students to incorporate more styles and design patterns, and to conduct experiments to meet the specified quality requirements.

Software life cycle: Students learn the importance of quality assurance in the early software life cycle to save cost and effort later on.

CONCLUSIONS AND FUTURE WORK

We have developed an architecture-based software reliability modeling tool to support the reliability measurement in the architectural design stage of the software life cycle. This tool can facilitate students to understand the importance of software architecture, and to discuss the quality tradeoffs among alternative architectural designs. The tool equips an easy-to-use GUI with informative display options on the intermediate reliability computation results, overcoming students’ past learning difficulties. With the support of this tool, students have felt much more easily and efficiently to exercise large-scale state models that combine with various architectural styles. The success of this tool is that many questions have become self-answered through students’ own repetitive practices. This strengthens their confidence and shortens the learning curve.

The support of this tool has also been a great help in teaching several other technical courses. Our next plan is to broaden its use to more courses in our software engineering curriculum. The future work is to develop a software environment that integrates this reliability modeling tool with another test coverage tool developed in the software testing course. Such integration will allow students to test a running system up to a good degree of coverage and the test results can then serve as the input to the reliability model to compute software reliability.

REFERENCES