Empir Software Eng (2016) 21:2413–2455
DOI 10.1007/s10664-015-9412-6

An automated software reliability prediction system for safety critical software

Xiang Li[1] · Chetan Mutha[2] · Carol S. Smidts[3]

Published online: 23 November 2015

Abstract Software reliability is one of the most important software quality indicators. It is concerned with the probability that the software can execute without any unintended behavior in a given environment. In previous research we developed the Reliability Prediction System (RePS) methodology to predict the reliability of safety critical software such as those used in the nuclear industry. A RePS methodology relates the software engineering measures to software reliability using various models, and it was found that RePS's using Extended Finite State Machine (EFSM) models and fault data collected through various software engineering measures possess the most satisfying prediction capability. In this research the EFSM-based RePS methodology is improved and implemented into a tool called Automated Reliability Prediction System (ARPS). The features of the ARPS tool are introduced with a simple case study. An experiment using human subjects was also conducted to evaluate the usability of the tool, and the results demonstrate that the ARPS tool can indeed help the analyst apply the EFSM-based RePS methodology with less number of errors and lower error criticality.

Communicated by: Nachiappan Nagappan

✉ Xiang Li
li.984@buckeyemail.osu.edu

Chetan Mutha
mutha.4@osu.edu

Carol S. Smidts
Smidts.1@osu.edu

[1] Nuclear Engineering Program at The Ohio State University, Columbus, OH, USA

[2] Oblon, McClelland, Maier & Neustadt, LLP, Washingston D.C Metro Area, USA

[3] Department of Mechanical and Aerospace Engineering at The Ohio State University, Columbus, OH, USA

Keywords Software reliability · Reliability modeling · Experimental validation · Operational profile · Finite state machine

1 Introduction

Software reliability is defined as "the ability of a system or component to perform its required functions under stated conditions for a specified period of time (IEEE 1990)". It is one of the most important indicators of software quality (ISO/IEC 2001). Much research has been dedicated to the evaluation of software reliability. This includes the various software reliability growth models (SRGM) (Musa J 1975; Huang C 2005; Huang C et al. 2007), the fault-seeding models (Mills H 1972; Walia and Carver 2008) and models relating software engineering measu res to reliability (Li and Smidts 2003; Pham 2007; Smidts et al. 2015; Huang and Liu 2013, 2015). In previous research we investigated the relationship between software engineering measures and software reliability (Smidts and Li 2004; Smidts et al. 2010; Li et al. 2013). The Reliability Prediction System (RePS) (Smidts and Li 2004; Smidts et al. 2010) methodology was developed which pairs software measures with associated models. Among the RePS's that have been investigated, it is found that the ones possess the highest predictive power should: 1) rely on software measures that can determine software defects (or software faults, used interchangeably in this paper) throughout the entire lifecycle of software development and 2) be paired with structural models of the software of interest. The advantage of structural models is that they allow the study of fault propagation, which is essential for correctly predicting software reliability. Eventually the Extended Finite State Machine (EFSM) model (Wang and Liu 1993) was selected among the possible structural models because the RePS based on EFSM was particularly promising. However, it was also found that the construction of the EFSM model is time-consuming and error-prone. Training and automation are two means to reduce the error rate, and this research focuses on the latter one.

The ARPS tool functions as follows: first it allows the user to load the original version of the software documentation (Software Requirement Specifications, SRS; Software Design Documents, SDD; and source code). These documents are called "original" because they contain uncovered defects that will be fixed in the next version of the software under study. Second the analyst uses the features provided by the tool to construct the EFSM model, which models the logic of the software. The resulting EFSM model is called the original EFSM model that contains the same uncovered defects from the SRS/SDD/code. After that, the analyst enters the defect information and the tool will automatically map the defects onto the original EFSM model, from which the modified EFSM model is generated. Eventually the analyst needs to enter information related to the Operational Profile (OP) (Chen et al. 2006) of the software and the tool will calculate the software reliability using the Execution, Infection and Propagation analysis derived from the PIE theory (Li et al. 2013; Voas 1992). Our model of the software is hierarchical: the SRS level is considered to be the top layer of this hierarchy; the SDD level is the middle layer and the code is the bottom layer. This is because functions and requirements that are initially defined in the SRS are usually elaborated in the SDD, and are eventually implemented at the code level. Similarly, the hierarchical model follows the same pattern. The lower the layer, the more details it contains.

A question that needs to be answered is whether the ARPS tool helps reduce the number of errors committed as well as low down the criticality of the errors. Thus an experiment to evaluate the usability of the ARPS tool was conducted. The experiment used undergraduate engineering students as research subjects. They were trained with our EFSM-based RePS methodology and the ARPS tool and their performance was evaluated at the end of the experiment. Their performance was analyzed using statistical approaches and our findings show that the ARPS tool indeed helps the subjects reduce both the number of errors and the error criticality.

Note that although the ARPS tool was originally inspired from safety critical software in the nuclear industry, its usage is not limited to safety critical systems. The major assumptions of our methodology, as discussed in Section 6.1, are that the number of defects should be small and that they should be distributed sparsely. Therefore, software systems that follow these assumptions can be analyzed using the ARPS tool. These systems include other critical systems such as ultra-high reliability and ultra-high availability systems.

2 Related Work

The main features of the ARPS tool are to model the software logic and evaluate software reliability. Thus this section on related work will review software reliability modeling techniques. Lyu M (1996); Smidts C, Li B et al. (2002); Pandey A and Goyal N (2013) provide a summary on software reliability models. In Smidts C, Li B et al. (2002), all models are roughly categorized based on 1) the phases of the software development life-cycle, 2) the information involved in the modeling, and 3) if the model requires the specific software structure or not. Typical modeling techniques for each categorization criterion are reviewed below.

Prediction models can be categorized into early-prediction and late-prediction models. Cheung L, Roshandel R et al (2008) describes an early prediction research study where a software component reliability prediction framework is developed by investigating architectural models. The work tries to address the problem of uncertainties associated with components under development, which is a major issue for existing architectural level reliability prediction approaches. Software reliability growth models (SRGM) (Musa J 1975; Huang C 2005; Huang C et al. 2007) fall into the late-prediction models' category. This is because these models are applicable to the software testing phases, where software faults are identified and removed dynamically. If it is assumed that the rate of introducing new faults during testing is lower than that of fault removal, software reliability increases as the testing progresses. In our case, because the EFSM-based RePS methodology uses SRS, SDD and code information, it does not really fall into early or late categories. In other words, if the EFSM-based RePS methodology only uses SRS or SDD information to predict software reliability, it is early-prediction; if it uses the code information (which involves testing information), it is a late-prediction approach. Gaffney G and Pietrolewiez J (1990) also discuss a phase-based model to predict software reliability. In their method the fault statistics obtained during the review of SRS, SDD and the coding phases are used, from which the software reliability is evaluated.

Based on the information involved in modeling, the prediction models can be categorized into failure-based models (e.g., the SRGMs introduced earlier), fault-based models

and development information-based models. One typical fault-based model is introduced by Mills (1972), where capture-recapture models that originated in biology are adopted to software engineering. In this study, an independent researcher seeds N_S faults which are designed to be representative of the indigenous faults into the software. Other researchers review or test the modified software and should identify seeded and indigenous faults. From the number of seeded and indigenous faults one can obtain an estimation of the number of faults remaining in the software. In our case the EFSM-based RePS methodology makes use of software fault information at the SRS, SDD and code level. However, our methodology only uses the indigenous faults already identified and hence significantly differs from the existing fault-based models. Bayesian Belief Networks (BBN) (Fenton and Neil 1999; Langseth and Portinale 2007) can be used as development information-based models. BBNs utilize a graphical network representation to illustrate the probability relationships among uncertain events. BBNs are all directed acyclic graphs where nodes represent random variables and edges represent conditional dependencies. For example, software reliability is represented as a random variable that connects to "number of latent faults" and "operational usage" (Fenton and Neil 1999). The software reliability can hence be evaluated using this relationship and the associated probability values.

Many prediction models can be categorized as black box models because they only rely on the number of faults or failures without considering the implementation of the software. On the other hand, some of the models require the software's structural information, and hence are categorized as architectural models. The early prediction research in (Cheung et al. 2008) falls into this category. In Gokhale and Trivedi (2006), the SRS level functional architecture is constructed and used as the system representation, from which the software failure probability is evaluated from the failure probability of each element. In our case the EFSM-based RePS methodology also requires the architectural information of the software which is used to generate the EFSM system representation. In addition, the defect information is mapped onto the EFSM model to illustrate the local effect of each defect. Therefore, one can consider that the EFSM-based RePS methodology belongs to the category of architectural models for software reliability modeling.

Tools have been developed to support software reliability prediction/assessment. For instance, (Lyu and Nikora 1992; Ramani et al. 2000; Chen et al. 2006) discuss tools implementing various SRGMs; Wang and Scannell (2005) discusses an architecture-based modeling tool and its support to teaching; Boudali and Dugan (2006) introduces a continuous-time BBN based framework. These tools are different from the ARPS tool because ARPS is based on PIE-based reliability assessment (Li et al. 2013; Voas 1992), which is not among the traditional software reliability modeling methodologies. In addition, although ARPS utilizes some of the architecture information, it represents the architecture using a hierarchical EFSM. These two characteristics differentiate ARPS from existing software reliability tools.

The following sections of this paper are organized as follows: Section 3 introduces the high level layout of the ARPS tool. Sections 4 to 9 elaborate the main features of the ARPS tool. Section 10 discusses our experimental validation for assessing the usability of the tool. The paper concludes in Section 11 and discusses future research.

3 Layout of the ARPS Tool

This section provides an overview of the ARPS tool. Detailed information can be found from Sections 4 to 9. Figure 1 shows the overall layout of the ARPS tool. Because the

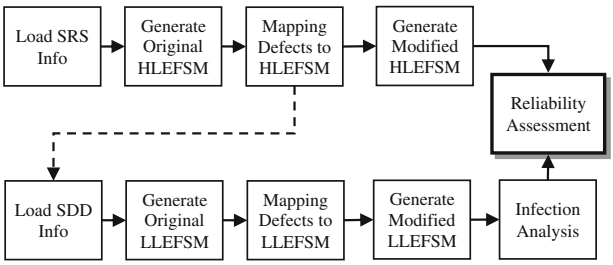


Fig. 1 Overall layout of the ARPS tool

implementation of the code level features has not yet been undertaken, only SRS and SDD level features are provided. From Fig. 1 it can be seen that the SRS level and the SDD level share some common characteristics. For a SRS level analysis, the process starts with loading the software under study's (SUS's) documents (the original SRS which contains defects and defect reports). The documents are used to guide the users to construct the original High Level EFSM (HLEFSM, or SRS Level EFSM), which is the graphical representation of the high level logic of the entire software. The documents include defect reports listing the location, category and a detailed description of the defects. In this research a set of typical defects were identified and templates to model the impact of these defects are defined (Li et al. 2013). These templates are implemented in the ARPS tool so that once a defect is pinpointed the corresponding template is applied automatically.

There are two branches coming out from "Mapping defects to HLEFSM" in Fig. 1. The top branch goes to "Generate Modified HLEFSM", where the original HLEFSM is modified to visualize the defects. The bottom branch that goes to "Load SDD Info" is shown in a dash line. This indicates the fact that the SDD information and the following analyses are not always necessary. As long as the defect is "large" enough to be modeled precisely at the SRS level, there is no need to further refine the model to the SDD level. However, from later sections in this paper it will be seen that some of the defects such as "Function with Incorrect Logic" cannot be appropriately modeled at the SRS level, and hence SDD level information must be factored in the model. In this case, the SDD information characterizing the defective state transition will be loaded into the ARPS tool. The user will then follow a procedure similar to the SRS level analysis to generate the original Low Level EFSM (LLEFSM, or SDD Level EFSM). This LLEFSM displays the local logic for the defective state transition or predicate in the HLEFSM. The defect that originates from the SRS level is decomposed into one or multiple SDD level defect(s). These SDD defects will be mapped onto the original LLEFSM using the corresponding templates. Similar to mapping the SRS defects, mapping the SDD defects leads to the modified LLEFSM, which illustrates the local effect of the SDD defects. These defects are also used to calculate the failure probability of this state transition (the box titled "Infection Analysis"), which will be fed back into the reliability calculation at the SRS level.

Our methodology is hierarchical. There are currently two major levels: the SRS level (i.e., high level) and the SDD level (i.e., low level). The SRS level is used to represent the logic of the entire program. There are two levels within the SRS level: level 0 and level 1 (see Fig. 2). The SRS level 0 only contains the SRS level 0 function which is the SUS itself; the SRS level 1 contains all the functions, variables and predicates that compose the SRS level 0 function. They are hence called SRS level 1 functions, SRS level 1 variables and SRS level 1 predicates, respectively. The SDD level is used to represent the specific

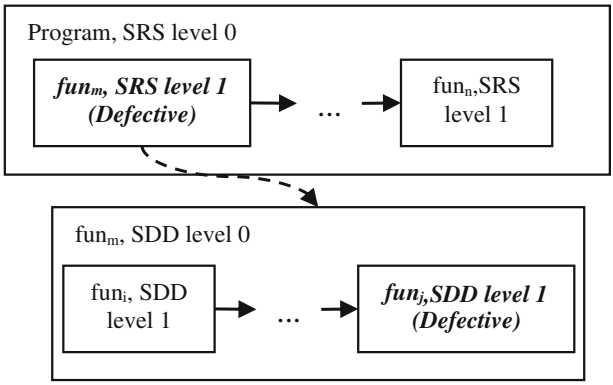


Fig. 2 Hierarchical levels in our methodology

logic of certain SRS level 1 functions, if they are (1) defective and (2) the defect cannot be precisely represented at the SRS level. Within the SDD level there are also two levels: level 0 and level 1. The SDD level 0 contains the SRS level 1 defective function that must be modeled at the SDD level. This function is also called the SDD level 0 function. The SDD level 1 contains the SDD functions, variables and predicates that constitute the SDD level 0 function. They are hence called SDD level 1 functions, SDD level 1 variables and SDD level 1 predicates, respectively.

4 Collecting Information About the Software Requirement Specification (SRS) and Constructing the Original HLEFSM

In this section the SRS level information collection and the HLEFSM construction are discussed. They correspond to the "Load SRS Info" and "Generate Original HLEFSM" blocks in Fig. 1. In the EFSM-based RePS methodology the logic of the software at the SRS and SDD level is visualized using the EFSM model. As is discussed in Li X, Li et al. (2013) and Wang and Liu (1993), "an EFSM in this study is a septuple (Σ, Γ, S, T, P, IV, OP) which is defined as follows:

- Σ is the set of input variables of the software. These variables cross the boundary of the software application.
- Γ is the set of output variables of the software. These variables cross the boundary of the software application.
- S is a finite, non-empty set of states. A state usually corresponds to the real-world condition of the system.
- T is the set of transitions. An event causes a state change and this state change is represented by a transition from one state to another.
- P is the set of predicates. The logical value of the predicates is attached to the related transition.
- IV is the set of internal variables defined and used within the software application boundary.
- OP is the set of probabilities of the variables that are used in the predicates. These variables can be from Σ, Γ or IV."

Software Requirement Specification for the Valve-Control System (VCS)

1. Introduction

… (Information ignored here)

2. Overall Description

The logic of the system is as follows: First the system is initialized based on temperature and pressure. If the temperature is greater than or equal to 45 ºC, valve V1 is opened, and then valve V2 is opened. Otherwise valve V3 is opened. At last post processing is conducted.

... (Information ignored here)

3. Specific Requirements

3.1 External Interface Requirements

… (Information ignored here)

3.2 Functional Requirements

3.2.1 Initialize the system: *(Information ignored)*

3.2.2 Open valve 1: *(Information ignored)*

3.2.3 Open valve 2: *(Information ignored)*

3.2.4 Open valve 3: open valve V3 based on the temperature

3.2.5 Post processing: conduct post processing tasks

… (Information ignored here)

Fig. 3 A snippet of the SRS for a simple valve-control system

In the above definitions, the inputs, outputs and internal variables are essentially all variables, but the inputs and outputs cross the boundary of the software application. This characteristic distinguishes them from the internal variables. Σ (inputs) and Γ (outputs) are user meaningful variables while the set IV is not. For convenience a set V can be defined which is the superset of Σ, Γ and IV, and some of the members of V are associated with the operational profile (OP). Note that both the correct and the defective elements of a program can be modeled using the above notation. In this section the features that are necessary for collecting information for all elements of the original SRS except for OP are discussed.

The features are introduced along with a case study SUS called the Valve-Control System (VCS) software. Figure 3 shows a snippet of the original SRS for VCS written in the format recommended by IEEE Std 830 (1998). In Fig. 3 only information regarding the logic of the software and a portion of the functional requirements is provided. The Overall Description section names the four functions which are further defined in the "Specific Requirements" section.

As mentioned earlier the program itself is defined as the SRS level 0 function in the HLEFSM. All functions defined in the specific requirements section are called SRS level 1 functions and each corresponds to one state transition. The predicate within the SRS level

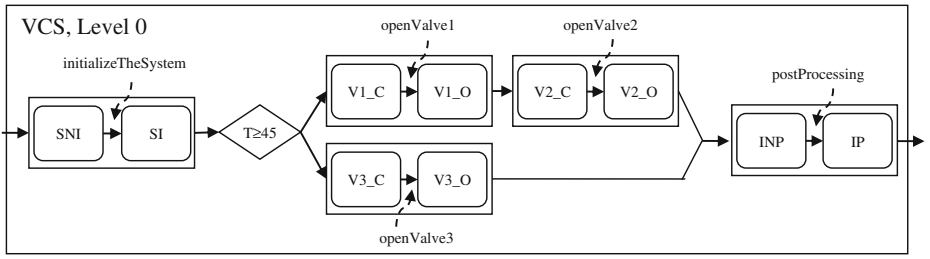


Fig. 4 The original HLEFSM corresponding to the SRS provided in Fig. 3

0 function is the SRS level 1 predicate. Thus VCS consists of four SRS level 1 state transitions and one predicate, and represent the same logic as provided in the plain text of the original SRS. VCS also has input variables "Temperature" and "Pressure" while no output is declared. Therefore, the user can gather the following information:

$\Sigma =$ Temperature, Pressure;
$\Gamma =$ Ø;
S = "System not initialized", "System initialized", "Valve1 closed", "Valve1 opened", "Valve2 closed", "Valve2 opened", "Valve3 closed", "Valve3 opened", "Information not post processed", "Information post processed";
T = initializeTheSystem, openValve1, openValve2, openValve3, postProcessing;
P = "Temperature $\geq$ 45";
IV = Ø.

OP is not used until the Execution, Infection and Propagation analysis are discussed. It should be noted that since this SRS is the original, it contains uncovered defects. Section 5 will discuss in detail how to handle these defects.

The HLEFSM in Fig. 4 displays the logic of VCS at the SRS level, which is built based upon the logic discussed in Fig. 3. Each function leads to a state transition which connects two states of S (note that in Fig. 4 all state names are acronyms). In our notation each state is bounded by a square with rounded corners and linked by an arrow. The name of the function that causes the state transition is written on the arrow. A predicate is a diamond with the condition written in the middle. In this case it is abbreviated as "T $\geq$ 45", where "T" is one of the input "Temperature".

Figure 5 displays a snapshot of the ARPS tool panel for collecting the original SRS information. The left side displays the original SRS, to which the analyst can refer, extract and record the relevant information into the tool. In the middle window one can see that the functions form a tree structure where VCS is at the top and all other functions are one level lower. The right side window shows the details of each function. A procedural language is introduced to allow logic entry at the SRS and SDD level. This language is called EFSM

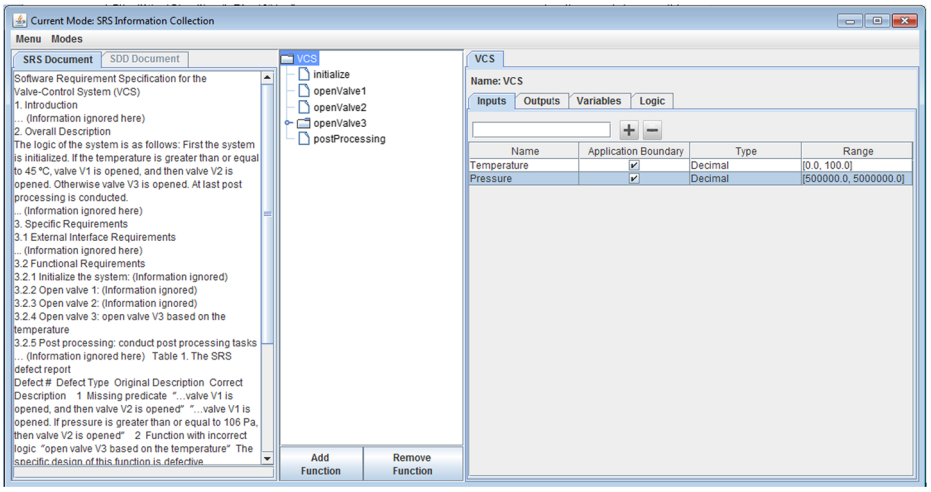


Fig. 5 Snapshot of the ARPS tool panel for collecting the original HLEFSM

language since one of its major missions is to help constructing the EFSM representation. Currently this language possesses the most typical binary and unary operators such as arithmetic operators, relational operators and so forth. It also supports function calls and different flow control statements such as the if-else structure and while loop. Because the language shares many features with the other popular procedural languages such as C, it is supposed to be easily understandable and usable by analysts with engineering background and average programming language knowledge. This is also a hypothesis that should be evaluated in our experiment. Once the logic of the original SRS is described using the EFSM language, a MATLAB (Mathworks 2014) script is generated and executed to draw the original HLEFSM. Figure 6 provides the snapshot of the MATLAB generated original HLEFSM for Fig. 4. The only difference is in the fact that the logic flow in Fig. 4 is from left to right while the logic flow in Fig. 6 is from top to down. In addition, this graphical representation is only for view but not for execution at this point. However, this could be part of future research.

5 Mapping SRS Defects to the Original HLEFSM

As discussed in Section 4, the HLEFSM constructed is the original and as such it contains uncovered defects. Currently 26 defects have been identified based on previous experience on software inspection and reliability modeling (Smidts et al. 2010). These 26 defect types

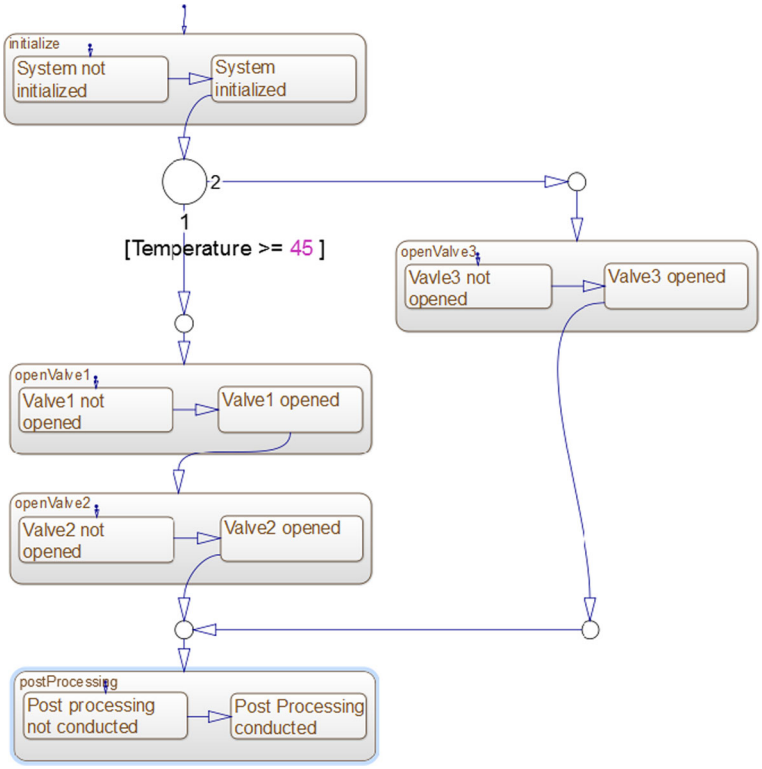


Fig. 6 Snapshot of the original HLEFSM automatically generated by MATALAB

can be categorized into 5 categories as shown in Table 1. Note that these terminologies are applicable to both SRS and SDD level, but in this section only the SRS level defects are discussed. Category 1 defects are about the definitions of the level 1 functions. If any of these

Table 1 SRS/SDD Defect studies in this research

Defect category	Defect name
Category 1: Defects for the definition of level 1 functions	1. Missing (definition of) Function: The entire definition of a function is missing from the SRS/SDD. 2. Extra (definition of) Function: The entire function definition is extra in the SRS/SDD. 3. Function with Incorrect Logic: The functionality of a function is valid but the logic is erroneous. 4. Incorrect Functionality: The functionality of the function is not valid. 5. Incorrect/Ambiguous Function Name: The name of the function is incorrect/ambiguous.
Category 2: Defects related to inputs	1. Missing Input: The definition of an input is missing from the SRS/SDD. 2. Extra Input: The definition of an input is extra in the SRS/SDD. 3. Incorrect/Ambiguous Input Name: The name of the input is incorrect or ambiguous when it is defined. 4. Input with Incorrect Type: The type of the input is erroneously defined. 5. Input with Incorrect Range: The range of the input is erroneously defined.
Category 3: Defects related to outputs	1. Missing Output: The definition of an output is missing from the SRS/SDD. 2. Extra Output: The definition of an output is extra in the SRS/SDD. 3. Incorrect/Ambiguous Output Name: The name of the output is incorrect or ambiguous when it is defined. 4. Output with Incorrect Type: The type of the output is erroneously defined. 5. Output with Incorrect Range: The range of the output is erroneously defined.
Category 4: Defects related to internal variables	1. Missing Variable: The definition of a variable is missing from the SRS/SDD. 2. Extra Variable: The definition of a variable is extra in the SRS/SDD. 3. Incorrect/Ambiguous Variable Name: The name of the variable is incorrect or ambiguous when it is defined. 4. Variable with Incorrect Type: The type of the variable is erroneously defined. 5. Variable with Incorrect Range: The range of the variable is erroneously defined.
Category 5: Defects for the logic of the level 0 function	1. Missing Instance of Function: The definition of a function is correct, but a call to that function is missing. 2. Extra Instance of Function: The definition of a function is correct, but there is an extra call to that function. 3. Incorrect/Ambiguous Function Call: The definition of a function is correct, but when it is called by another function, the name is incorrect or ambiguous. 4. Missing Predicate: A predicate of the function is missing. 5. Extra Predicate: A predicate of the function is extra. 6. Incorrect/Ambiguous Predicate: The logical expression of a predicate is incorrect or ambiguous.

Table 2 SRS level defect report for the VCS exaple

Defect #	Defect Type	Original Description	Correct Description
1	Missing Predicate	"... valve V1 is opened, and then valve V2 is opened"	"... valve V1 is opened. If pressure is greater than or equal to 10^6 Pa then valve V2 is opened"
2	Function with Incorrect Logic	"open valve 3 based on the temperature"	The specific design of this function is defective.

defects exist, they will affect all instances of that function. Categories 2 to 4 are about the inputs, outputs and internal variables. In our methodology they all have the same properties: name, type, range and whether they cross the application boundary. Thus the defect types are similar. It should be noted that inputs, outputs are both for the level 0 function and the level 1 functions, while the internal variables are only for the level 0 function. The internal details of each level 1 function (which include the internal variables and logic) are only discussed when necessary, i.e., there exist defects in this level 1 function and they cannot be precisely represented at the SRS level. This is because our methodology relies on an important underlying philosophy, i.e. details of level 1 functions should remain hidden as long as their exploration is unnecessary. This underlying philosophy helps save significant modeling time and computational resources. Sections 7 and 9 will discuss how to handle SDD defects. Category 5 defects are concerned with the logic of the level 0 function. Also Defect #1 of Category 5 should not be confused with Defect #1 of Category 1. While Category 1 defects apply to all instances of a function, Category 5 defects apply to a single instance of this function.

The defect categorization discussed above originates from the software failure mode taxonomy discussed in Li et al. (2005). In this reference the software failure modes are categorized based on software functions and their interactions, which are investigated to identify where/how software failures may occur. The function, attribute and function set failures in Table 1 of the reference correspond to the Category 1 and 5 defects in our categorization. The input and output failure modes in Table 2 of the reference correspond to Category 2 and 3 defects in our categorization. Our Category 4 defects are extensions of our Category 2 and 3 defects. In Li et al., the failure mode taxonomy was also validated using both subjective evidence, i.e., expert judgment, and, objective evidence, i.e., actual failure mode data. The validation process shows that the taxonomy is applicable to the domain of aerospace applications, which is one of the major categories of safety critical systems. In reference Smidts et al. (2010) a similar defect categorization as the one in this paper is proposed, where the

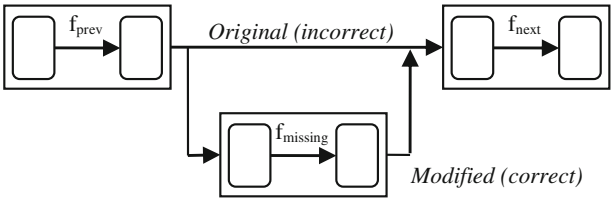


Fig. 7 Defect template for "Missing Instance of Function"

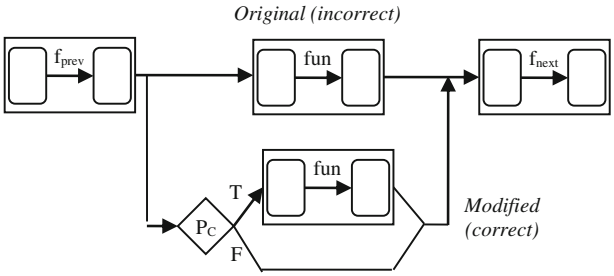


Fig. 8 Defect template for "Missing Predicate"

possibility of using EFSM as a software logic representation and mapping software defects to the EFSM model to evaluate software reliability is discussed.

A template allowing us to map each defect type to the original HLEFSM (thus there are 26 templates) has been introduced. Figures 7 and 8 provide the templates for "Missing Instance of Function" and "Missing Predicate", respectively. It can be seen that there are two branches in the middle portion of each Figure. The upper branch is the original branch that contains the defect; the lower branch is the modified branch which displays the corrected representation. Therefore, for "Missing Instance of Function" in Fig. 7 the upper branch is just an arrow linking the previous state transition block "f_{prev}" with the next state transition block "f_{next}". There is no state transition block in the middle. However, the lower branch contains the missing state transition block "$f_{missing}$" and hence is correct. Similarly, in Fig. 8 the upper branch is just a state transition block "fun" and the predicate that should have been evaluated before "fun" is missing. In the lower branch one can see that the missing predicate "P_C" is added, and when "P_C" is evaluated as false (F), no state transition occurs.

The defects' templates can be applied to the VCS example in Fig. 3 to illustrate how mapping defects is performed. Suppose a defect report such as the one provided in Table 2

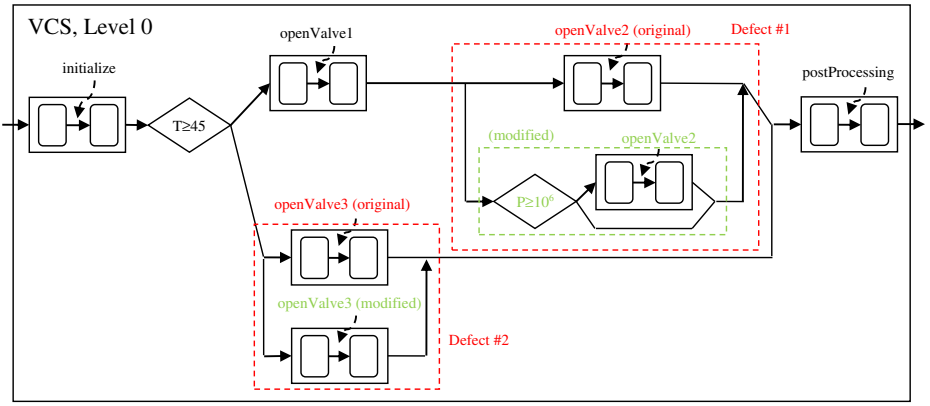


Fig. 9 The modified HLEFSM corresponding to the defect report provided in Table 2

is available. Two defects exist: one is a "Missing Predicate" and the other is a "Function with Incorrect Logic". (It should be noted that this defect report is designed for the sake of demonstrating how multiple defects are mapped to the HLEFSM. In reality there should not be so many defects in such a small system, especially not in a safety critical system.) Fig. 9 is the modified HLEFSM with both defects mapped, where each defective fragment is delineated by a dotted line. "T $\geq$ 45" is short for "Temperature $\geq$ 45" and "P $\geq 10^6$" is short for "Pressure $\geq 10^6$". Similar as the templates provided in Figs. 7 and 8, the upper branch is the original (incorrect) version and the lower branch is the modified (correct) version. Defect #1 is mapped in the same way as was shown in Fig. 8. Defect #2 is about the inner logic of the function "openValve3", and the template for "Function with Incorrect Logic" follows the same style as the other templates. However, for this type of defect it is very common that the SRS level information is insufficient for conducting the Infection analysis (see Section 6). Therefore, the user will need to delve into the SDD level to model the defect more precisely. When using the ARPS tool to map defects, the EFSM language is used to capture the correct functions/predicates and MATLAB is called to generate the modified HLEFSM.

6 Execution, Infection and Propagation Analysis for Software Reliability Evaluation

6.1 Formula for Software Reliability Calculation

The modified HLEFSM is helpful in visualizing the logic of the SRS. In addition, it can serve as a system representation on which the Execution, Infection and Propagation analysis for reliability evaluation can be carried out. In the EFSM-based RePS methodology, a software fault (i.e., defect) causes a software failure only when three constraints are satisfied. First the defect must be executed (i.e., Execution). Second the execution of the defect must make a difference in the data state of the software (i.e., Infection). Third the data state difference must be preserved and propagated to the end of the software and hence visible from the outside (i.e., Propagation). The Execution, Infection and Propagation analysis are introduced to assess the probability that the three constraints are satisfied for each defect. Once the analysis is conducted for all defects, (1) and (2) can be applied to obtain software reliability. Previous research has shown that these two equations are valid when the total number of defects in software is very small and the defects are distributed sparsely (Smidts et al. 2010; Li et al. 2013; Voas 1992). These conditions are reasonable for safety critical software which has undergone thorough software testing, software verification and validation processes.

$$Prob\,(failure) = \sum_{i=0}^{N} E_i \times I_i \times P_i \tag{1}$$

$$Re = 1 - Prob\,(failure) \tag{2}$$

where in the above formula,

E_i: The execution probability for the *i-th* uncovered fault
I_i: The infection probability for the *i-th* uncovered fault
P_i: The propagation probability for the *i-th* uncovered fault

N: The total number of uncovered faults
Prob(failure): Probability of failure
Re: Software reliability

The Execution, Infection and Propagation analysis all require the operational profile (OP) of the software. The operational profile is defined as a set of probabilities characterizing the environment within which a software system is to carry out its mission (Musa et al. 1987; Musa 1993). In this study it is assumed that the user of ARPS has access to the OP data.

6.2 Execution Analysis

The Execution analysis identifies the probability that each defect is executed. This analysis is conducted during the compile time of our EFSM language. More specifically, an inherited attribute (Lam et al. 2006) "Precondition" is introduced for each statement, which consists of all the predicates passed down from the parent node in the parse tree. Assuming there are N predicates, the "Precondition" can be expressed as:

$$Precondition = P_1\left(v_{P_{11}}, v_{P_{12}}, \ldots\right) \wedge P_2\left(v_{P_{21}}, v_{P_{22}}, \ldots\right) \wedge \ldots \wedge P_N(v_{P_{N1}} v_{P_{N2}}, \ldots v_{P_{NM}})$$

Where each predicate $P_i$can be either true or false, and $v_{P_{NM}}$ is the last variable of P_N. The Execution probability is then equal to $Prob(Precondition)$. If all v_P variables are input variables (i.e., they belong to I) and the joint probability distribution function (pdf) $f_{V_{P_{11}}, V_{P_{12}}, \ldots V_{P_{NM}}}(v_{P_{11}} v_{P_{12}}, \ldots v_{P_{NM}})$ is available from the OP, the Execution probability can be calculated as:

$$Prob\ (Precondition) = \int_{V_{P_{11}} \in \left[v_{P_{11,min}}, v_{P_{11,max}}\right], \ldots V_{P_{NM}} \in \left[v_{P_{11,min}}, v_{P_{11,max}}\right]} f_{V_{P_{11}}, V_{P_{12}}, \ldots V_{P_{NM}}}(v_{P_{11}}, v_{P_{12}}, \ldots v_{P_{NM}}) dv_{P_{11}} \ldots dv_{P_{NM}} \tag{3}$$

In formula (3), the domain on which the predicates hold (i.e., $\left[v_{P_{11,min}}, v_{P_{11,max}}\right]$, $\left[v_{P_{12,min}}, v_{P_{12,max}}\right]$, etc.) can be calculated by calling a symbolic solver (e.g., Mathematica (Wolfram 2014) and MATLAB (Mathworks 2014)). Then the OP is used to calculate the probability by solving the multiple integral (3).

If some of the v_P variables are internal variables of the program, their pdfs are unavailable from OP. However, the relationship between the internal variables and the input variables can be examined based on the program logic. Because these variables are at the SRS level, the relationship should be straightforward. Assume the input variables of a program are x and y, and the variables of a predicate are u and v. The relationship between the variables is as follows:

$$\begin{cases} x = x(u, v) \\ y = y(u, v) \end{cases} or \begin{cases} u = u(x, y) \\ v = v(x, y) \end{cases}$$

Thus the pdf for the random vector (U, V) can be obtained from that of (X, Y):

$$f_{U,V}(u, v) = f_{X,Y}(x(uv), y(uv)) \left| \begin{array}{cc} \frac{\partial x}{\partial} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{array} \right|$$

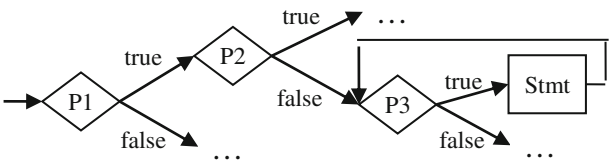


Fig. 10 Preconditions of the statement "Stmt"

Where the last term is the Jacobian determinant. This formula can be inserted into (3) to calculate Prob(Precondition). For random vectors with more than 2 variables, the formula can be extended. Now let us consider an example. The statement "Stmt" in Fig. 10 is executed when: 1) the predicate "P1" evaluates as true and 2) "P2" evaluates as false and 3) "P3" evaluates as true. Thus the "Precondition" of "Stmt" can be expressed as (P1==true $\bigwedge$ P2==false $\bigwedge$ P3==true). The problem of Execution analysis for "Stmt" is hence to find out the value of Prob(P1==true $\bigwedge$ P2==false $\bigwedge$ P3==true).

Assume x, y and z are the input variables involved in each predicate, respectively. Let X, Y and Z be the corresponding random variables. The "Precondition" holds when X ϵ [x1, x2], Y ϵ [y1, y2] and Z ϵ [z1, z2]. Also assume that the joint pdf is $f_{X,Y,Z}(x,y,z)$, the Execution probability is thus:

$$Prob\left(P_{1,X}\left(x\right) == truele \wedge P_{2,Y}\left(y\right) == false \wedge P_{3,Z}\left(z\right) == true\right) = \int_{X\in[x1,x2],Y\in[y1,y2],Z\in[z1,z2]} f_{X,Y,Z}(x,y,z)dxdydz$$

For the VCS example in Fig. 3, only the input variable Temperature is used in the predicate. The OP for Temperature is listed in Table 3. The pdf of Temperature is a truncated normal distribution. For Defect #1 the Execution probability is given by Prob(T $\geq$ 45), which is equal to 84.1 %; for Defect #2 the Execution Probability is given by Prob(T<45), which is equal to 15.9 %.

From Fig. 10 it can be seen that there is a back edge from "Stmt" to "P3", which means that "P3" is the predicate of a loop structure. In the EFSM-based RePS methodology these predicates are treated in the same way as the predicates of "if-else" structures. i.e., if a statement is within a loop body like "Stmt", its "Precondition" should be the conjunction of the "Precondition" of the parent node "P3" and the loop predicate.

Table 3 Operational profile of the VCS example

Key Input Variable	Probability Distribution Function
Temperature, Temp in °C	$f_{Temp}\left(temp\right) = \frac{1}{\sqrt{2\pi}\sigma_{temp}} \exp[-\frac{(temp-\mu_{temp})^2}{2\sigma_{temp}^2}]$ where $temp \in [0, 100]$, $\sigma_{temp} = 5$, $\mu_{temp} = 50$
Pressure, Pres in Pascal	$f_{Pres}\left(pres\right) = \frac{2}{2025\times10^{10}} pres - \frac{2}{405\times10^5}$ where $pres \in [5 \times 10^5, 5 \times 10^6]$

6.3 Infection Analysis

The Infection analysis identifies the probability that the defect causes a difference in the data state that immediately follows the defective block. Because the data state difference studied is local, the analysis is specific to each type of defect in Table 1. First let us only consider SRS level defects. Our study indicates that for Category 5 defects the Infection analysis can be conducted at the SRS level while for the other four categories the analysis should be conducted at the SDD level. This stems from the fact that for Category 5 defects the information at the SRS level is sufficient to determine the Infection probability, but for the other categories of defects this is usually not true. In this section only Category 5 defects are discussed. The other four categories are discussed in Section 7 and 8.

The six defects of Category 5 can be further classified as 1) function call related and 2) predicate related. Defect #1, #2 and #3 are function call related, and in all cases a wrong function call takes place. In our methodology it is assumed that a function call to an incorrect function always leads to a local data state difference, therefore the Infection probability linked to these three defects conservatively assumed to be 100 %, although other values can be specified by the user if need be. This conservative assumption is called "incorrect function assumption". It holds when each functionality (defective and correct) is significantly different from the others. This is usually the case for SRS level 1 functions because they are at a high level of abstraction.

Defect #4, #5 and #6 in Category 5 are predicate-related. Based on the "incorrect function assumption", the Infection probability related to Defects #4, #5 and #6 can be determined as well. Defect #6, Incorrect/Ambiguous Predicate, is the most general case among these three. Figure 11 provides the defect template for Incorrect/Ambiguous Predicate. The P_O in the upper branch is the original (i.e., incorrect) predicate, and P_C in the lower branch is the modified (i.e., correct) predicate. Defect #4 and #5 are two special cases of Incorrect/Ambiguous Predicate: for Missing Predicate, P_O is a predicate always evaluating to true (Fig. 8); for Extra Predicate, one can imagine that there is a predicate P_C (not shown) on the modified branch which is always evaluated to true (Fig. 12).

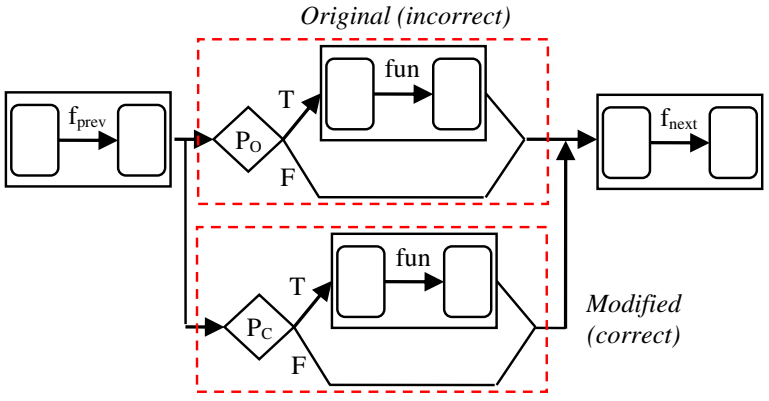


Fig. 11 Defect template for "Incorrect/Ambiguous Predicate"

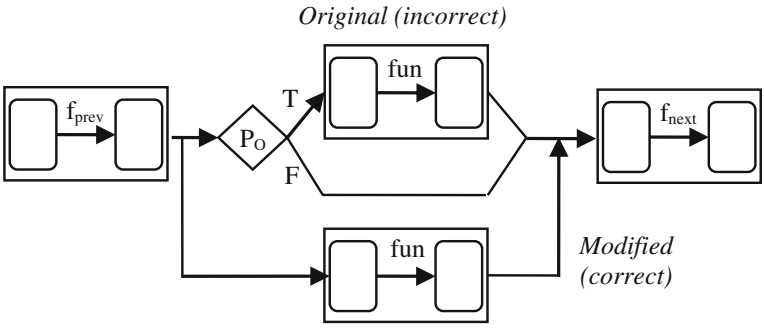


Fig. 12 Defect template for "Extra Predicate"

The Infection analysis consists in assessing the probability $Prob\,(P_O\ is\ incorrect)$. First let us focus on an Incorrect/Ambiguous Predicate. The function call to "fun" remains the same in both branches (see Fig. 11) while the predicate in each branch is different. The event "P_O is incorrect" can be decomposed based on P_C, as shown in (4).

$$Prob\,(P_O\ is\ incorrect) = Prob((P_O\ is\ incorrect \cap P_C) \cup (P_O\ is\ incorrect \cap \overline{P_C})) \quad (4)$$

For the term $(P_O\ is\ incorrect \cap P_C)$, when P_C is true, the event "P_O is incorrect" corresponds to P_O being false. This is because when P_O is false, the wrong branch is taken. Hence this term can be written as $\bar{P_O} \cap P_C$. For the term $(P_O\ is\ incorrect \cap \bar{P_C})$, when P_C is false, the event "P_O is incorrect" corresponds to P_O being true. This is again because when P_O is true, the wrong branch is taken. Hence this term can be written as $P_O \cap \bar{P_C}$. Therefore, (4) can be further rewritten as (5).

$$I_{D\#6} = Prob\,(P_O\ is\ incorrect) = Prob((\overline{P_O} \cap P_C) \cup (P_O \cap \overline{P_C})) \quad (5)$$

This equation is also applicable to Defect #4 and #5. More specifically, for a Missing Predicate the original (incorrect) branch only contains a function call to "fun" (Fig. 8). Thus $P_O = \Omega$ and $\bar{P_O} = \emptyset$ (the universal set). Equation (5) can be modified as:

$$I_{D\#4} = Prob\left((\emptyset \cap P_C) \cup (\Omega \cap \overline{P_C})\right) = Pro(\overline{P_C}) \quad (6)$$

which means that the Infection probability for Missing Predicate is equal to the probability that P_C is false.

To apply (5) to an Extra Predicate (Fig. 12), an imaginary predicate P_C can be added onto the modified (correct) branch before "fun". It is always evaluated as true and thus $P_C = \Omega$ and $\bar{P_C} = \emptyset$. Therefore, (5) can be modified as:

$$I_{D\#5} = Prob\left((\overline{P_O} \cap \Omega) \cup (P_O \cap \emptyset)\right) = Prob(\overline{P_O}) \quad (7)$$

Equations (5) to (7) can be calculated if the OP of the related variables is available. The specific process is similar to the Execution analysis, where symbolic solvers may be needed to compute the domain on which the predicates hold. For the VCS example, Defect #1 (Fig. 9) falls into the category of Missing Predicate. Applying (6) it is found that $Prob(\overline{P_C})$ is equal to $Prob(\neg(Pres \geq 10^6))$. Based on Table 3 this probability is equal to 1.2 %.

The other four categories of defects are different from Category 5. The Infection analysis for Defects #1 and #2 of Category 1 depends on whether these two defects actually lead to erroneous instantiations. If they do, the Infection probability of the affected functions is

also 100 % as for Defect #1 and #2 of Category 5. For the other Category 1 defects and all Category 2, 3 and 4 defects the analysis usually needs to be conducted at the SDD level, since there isn't sufficient information at the SRS level. This topic will be further discussed in Sections 7 and 8. Table 4 summarizes the Infection analysis methodology for each defect category.

6.4 Propagation Analysis

The Propagation analysis is one of the most challenging topics in this research. How the data state transforms when passing through the downstream state transitions and predicates should be considered. Currently the theoretical portion of the Propagation analysis for defects affecting functions with one input variable has been completed and is described in a companion paper. The Propagation analysis is currently not implemented into the ARPS tool and Propagation probabilities are currently set to 100 % by default. However, the user is allowed to provide values other than 100 %. For the VCS example in Fig. 9, the Propagation probability for both SRS level defects is set to 10 %.

7 SDD Level Analysis I: Constructing the Original LLEFSM Using the SDD Information

A general philosophy of the EFSM-based RePS methodology is that the analysis should stay at the SRS level (the highest level) which contains the minimal amount of information about the software system. Performing analysis at the highest level and hiding lower level information will significantly reduce the modeling effort. However, as has been discussed in Section 6.3, occasionally the information at the SRS level is not detailed enough to conduct the Infection analysis. This is usually the case for the SRS level 1 functions. Because the SDD level information is more detailed, this information can be loaded to better model the defective functions. Defects #3, #4 and #5 of Category 1 and all defects of Category 2, 3 and 4 in Table 1 are introduced for this purpose.

Currently the SDD level modeling bears many similarities with the SRS level modeling. First, the SDD information about a certain SRS level 1 function is loaded. The analyst will

Table 4 Summary of the Infection analysis methodology

Defect Category	Infection Methodology
Category 1	Defect #1 and #2: If the defective function is instantiated, apply the "incorrect function assumption" Defect #3 to #5: Apply the SDD level analysis.
Category 2 to 4	All defects need to be analyzed at the SDD level.
Category 5	Defect #1 to #3: Apply the "incorrect function assumption" Defect #4: Calculate $Prob(\bar{P_C})$ Defect #5: Calculate $Prob(\bar{P_O})$ Defect #6: Calculate $Prob((\bar{P_O} \cap P_C) \cup (P_O \cap \bar{P_C}))$

Software Design Document for the Open Valve #3 function of the Valve-Control System (VCS)

1. Introduction to the Open Valve #3 function

… (Information ignored here)

2. System Overview

The logic of this function is as follows: First the current position of Valve #3 is obtained. If the position is below 2 mm, then Valve #3 should be rotated clockwise. Otherwise it should be rotated counter clockwise.

… (Information ignored here)

3. System Architecture

3.1 External Interface Requirements

… (Information ignored here)

3.2 Functional Requirements

3.2.1 Obtain the current valve position: *(Information ignored)*

3.2.2 Rotate the valve clockwise: *(Information ignored)*

3.2.3 Rotate the valve counter clockwise: *(Information ignored)*

… (Information ignored here)

Fig. 13 A snippet of the SDD for the "openValve3" function

be working with a panel whose layout is identical to the one displayed in Fig. 5. When a certain SRS level 1 function is analyzed at the SDD level, this function becomes the SDD level 0 function. It has inputs, outputs, internal variables, state transitions that are caused by SDD level 1 functions and SDD level predicates. The SDD level 1 functions also have inputs and outputs but do not have internal variables, state transitions or predicates. All defects listed in Table 1 exist at the SDD level, but the "level 0 function" and "level 1 function" should now be interpreted as "SDD level 0 function" and "SDD level 1 function", respectively. Note that at this point the SDD level 1 defects cannot be modeled automatically because the code level features have not been implemented. A manual analysis example is provided in Section 9. This topic will be further elaborated in Section 11, Conclusions and Future Research.

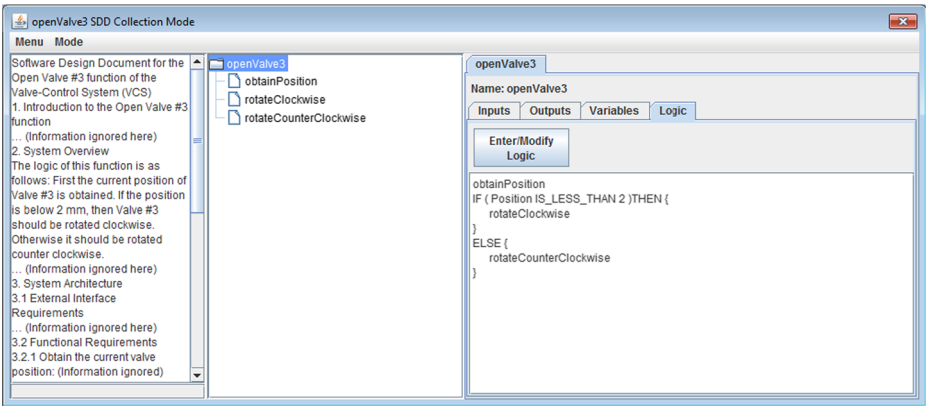


Fig. 14 Snapshot of the ARPS panel for entering the SDD information for "openValve3"

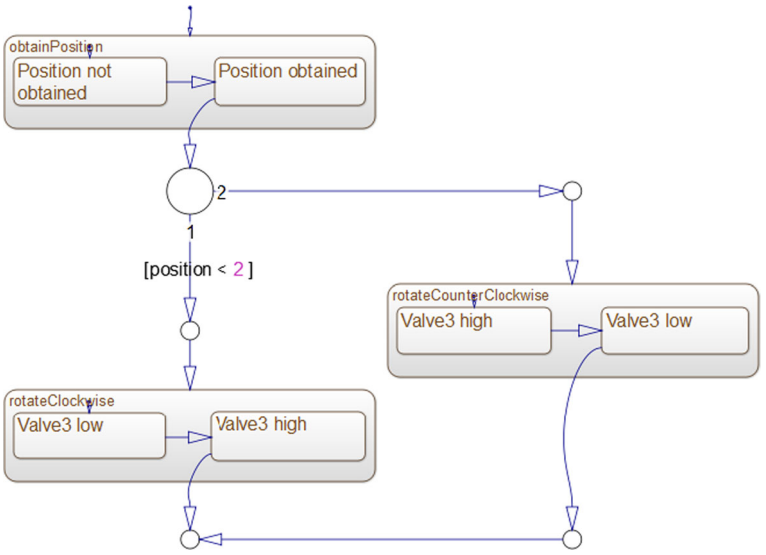


Fig. 15 Snapshot of the original LLEFSM generated by MATALB

As an example, Defect #2 for "openValve3" in Table 2 falls into the category of "Function with Incorrect Logic" and can be decomposed and analyzed at the SDD level. Assume that the original (i.e., defective) SDD in Fig. 13 is developed based on the SDD template provided in IEEE Std 1016 (1998). The function logic is explained in System Overview. The ARPS tool allows the user to load the original SDD, enter the relevant information and construct the original Low Level EFSM (LLEFSM). The specific procedure is the same as the SRS level information recording and HLEFSM construction. Occasionally the SDD is developed following other formats. For instance, it may consist mostly of diagrams or is written in UML (Booch et al. 2005). This will not cause fundamental differences in using the ARPS tool because the analyst can still view the SDD file from the tool panel and gather useful information.

The SDD document for "openValve3" is first loaded. Then the SDD level 1 functions for "openValve3" are specified by first clicking on the function name and then the "Adding Function" button (see Fig. 5). The ARPS panel will change to Fig. 14. The detailed SDD

Table 5 SDD level defect report for "openValve3"

Defect #	Defect Type	Original Description	Correct Descriptio
1	Incorrect/ Ambiguous Predicate	". ..If the position is below 2 mm, then Valve #3 should be rotated clockwise. Otherwise it should be rotated counter clockwise."	". If the position is below 3 mm, then Valve #3 should be rotated clockwise. Otherwise it should be rotated counter clockwise."

Table 6 SDD level OP for "openValve3"

Key Input Variable	Probability Distribution Function
Position, in mm	The relationship between Pressure and Position is: $Pressure = 5 \times 10^5 \times Position$ Thus the pdf of Position is: $f_{POS}(pos) = \frac{2}{81} pos - \frac{2}{81}$ where $pos \in [1, 10]$

information for "openValve3" is still displayed on the left hand side. The middle portion is the SDD level function tree. The right hand side displays the logic of the original (defective) "openValve3", which needs to be added by the user following the EFSM language syntax. Figure 15 displays the original LLEFSM generated by MATLAB following the EFSM language commands.

8 SDD Level Analysis II: Mapping SDD Defects to the Original LLEFSM and Performing the Infection Analysis

8.1 Mapping Defect #2 to the Original LLEFSM and Performing the Infection Analysis

Because the SRS level Execution, Infection and Propagation analysis are introduced for failure probability analysis, they are applicable to the SDD level as well. The resulting SDD level failure probability is hence the Infection probability of the same function at the SRS level. Mathematically, this relationship can be expressed as follows:

$$I_{fun_i,SRS} = Prob_{fun_i,SDD}(failure) = \sum_{j=0}^{M} E_{j,SDD} \times I_{j,SDD} \times P_{j,SDD} \quad (8)$$

where M is the total number of SDD level defects for function fun_i.

The specific Execution, Infection and Propagation analysis are the same as Section 6 and hence are not repeated. The SDD level defect reports and OP are still needed for the analysis. For function "openValve3" discussed in Figs. 13 and 15, the SDD level defect report says that there is a defect of type "Incorrect/Ambiguous Predicate". The defect report and the OP table are provided in Tables 5 and 6, respectively. There is a linear relationship between the internal variable "Position" and the input variable "Pressure", from which the pdf of "Position" can be derived. In addition, since there is no SDD level 1 function after the "if" structure, the Propagation probability of Defect #1 is simply 100 %. Applying the defect template in Fig. 11 as well as (1) the failure probability (i.e., the SRS level Infection probability) of "openValve3" is calculated as follows:

$$\begin{aligned} &Prob_{openValve3,SRS}(failure) = E_{1,SDD} \times I_{1,SDD} \times P_{1,SDD} \\ &= 100\ \% \times (Prob(\textit{ Position is greater than 2 and Position is less than 3 })) \times 100\ \% \\ &= 3.7\ \% \end{aligned}$$

The Execution, Infection and Propagation probabilities of the two SRS level defects of the VCS example have been obtained. Thus the software reliability can be assessed as follows:

$$Prob_{VCS}(failure) = E_{1,SRS} \times I_{1,SRS} \times P_{1,SRS} + E_{2,SRS} \times I_{2,SRS} \times P_{2,SRS}$$
$$= 85.1\,\% \times 1.2\,\% \times 10\,\% + 14.9\,\% \times 3.7\,\% \times 10\,\% = 0.16\,\%$$
$$Re_{VCS} = 1 - Prob_{VCS}(failure) = 99.84\,\%$$

The above analysis can be automatically conducted with the assistance of the ARPS tool. The "Mode" tab at the top-left corner of Fig. 14 is used to switch the mode between the "information collection mode" and the "defect collection mode". Once in the "defect collection mode", the user can further specify how exactly "openValve3" is defective based on the defect report. Since it is an "Incorrect/Ambiguous Predicate" at the SDD level, the logic panel should be selected and the modified logic should be entered. Figure 16 displays the SDD level logic panel with the modified EFSM language command. Comparing to the original logic on the right hand side in Fig. 14, we can see that the differences are the tags "<IAP_BEGIN>", "[PC=...]", "[PROPAGATION = ...]" and "<IAP_END>". These tags fully specify that the incorrect of "Position less than 2", and the correct version should be ?Position less than 3?. Once this information is provided, the failure probability of "openValve3" (or Infection probability at the SRS level) can be automatically calculated as 3.7 %.

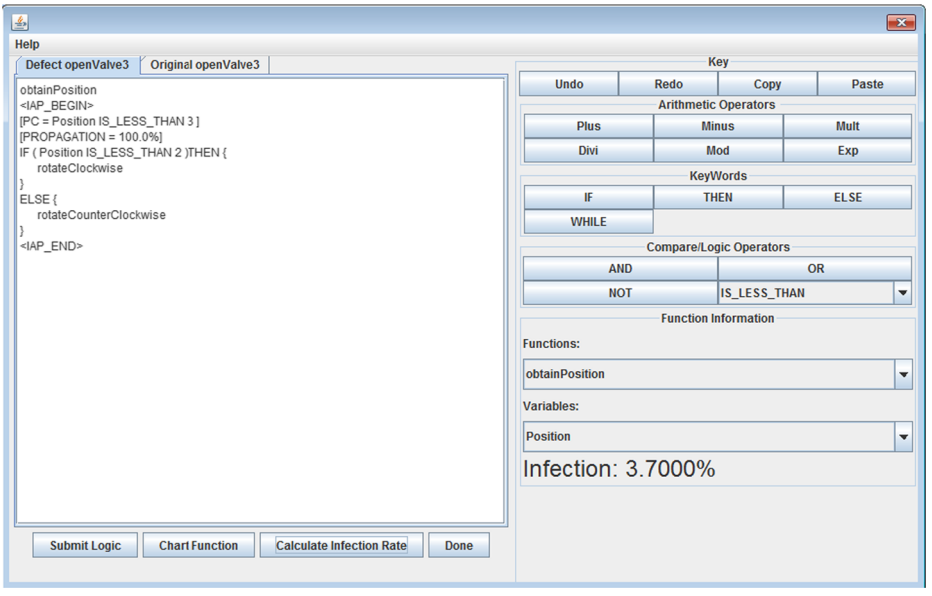


Fig. 16 Snapshot of the ARPS panel for specifying the range of the internal variable "Position"

8.2 Another Example to Demonstrate the SRS-SDD Defect Mapping and Failure Probability Assessment

So far Section 7 and 8.1 demonstrate how to analyze the SRS level defect of type "Function with Incorrect/Ambiguous Logic" at the SDD level. A similar principle can be applied to other types of defects that cannot be processed at the SRS level. For instance, Category 4 defects that are about SRS level internal variables usually need to be modeled at the SDD level. Assume that the SRS level defect on "openValve3" is "Variable with Incorrect Range" instead of "Function with Incorrect Logic", and the defective internal variable is "Position" The inner logic of "openValve3" is correct. Figure 17 displays the snapshot of the ARPS panel where the analyst specifies the range of "Position". Here the analyst enters the original (incorrect) range of Position which is [1, 9]. Figure 18 displays how the analyst can specify the correct range for "Position" He/she needs to switch to defect collection mode and enter the correct range of [1, 10] based on the new SDD level defect report provided in Table 7

As provided in Table 6, the correct pdf of "Position" is $f_{POS}(pos) = 2 / 81 * pos - 2 / 81$ and is distributed on [1, 10]. From the correct description of Table 6 we also know that "Position" is involved in the predicate "Position is below 3 mm". Because the incorrect range is [1, 9], this predicate will be affected. The likelihood of Position being in (9, 10] will be suppressed because of the boundary checking mechanisms. Thus the failure probability (i.e., the SRS level Infection probability) of "openValve3" is calculated as follows:

$$
\begin{aligned}
Prob'_{openValve3,SRS}(failure) &= E'_{1,SDD} \times I'_{1,SDD} \times P'_{1,SDD} \\
&= 100\,\% \times (Prob(\textit{ Position is greater than 9 and Position is less than equal to 10 })) \times 100\,\% \\
&= 20.1\,\%
\end{aligned}
$$

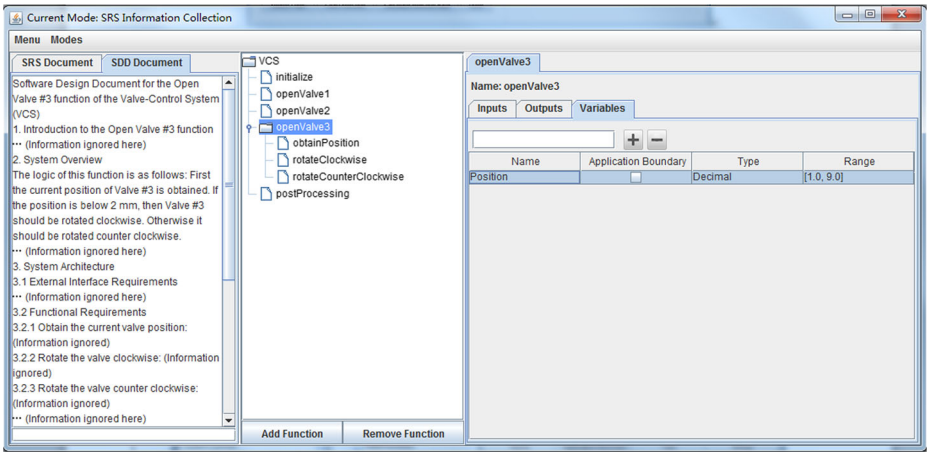


Fig. 17 Snapshot of the ARPS panel for specifying the range of the internal variable "Position"

Table 7 SDD level defect report for "openValve3"

Defect #	Defect Typ	Original Description	Correct Descriptio
1	Variable with Incorrect Range	The range of the internal variable "Position" is [1, 9]	The range of the internal variable "Position" is [1, 10]

And the software reliability can be assessed as follows:

$$Prob'_{VCS}(failure) = E_{1,SRS} \times I_{1,SRS} \times P_{1,SRS} + E'_{2,SRS} \times I'_{2,SRS} \times P'_{2,SRS}$$
$$= 85.1\,\% \times 1.2\,\% \times 10\,\% + 14.9\,\% \times 20.1\,\% \times 10\,\% = 0.4\,\%$$
$$Re'_{VCS} = 1 - Prob'_{VCS}(failure) = 99.6\,\%$$

9 Code Level Analysis

The code level automatic analysis features are still under development. Thus in this section a SDD-code level defect mapping example is provided, which is conducted manually. Assume that in addition to the two defects in the VCS case study that are already discussed, there is one more defect at the SRS level 1 function "postProcessing". This defect is of type "Function with incorrect logic", which has to be elaborated at the SDD level (see the left hand side to the middle of Fig. 19). However, unfortunately this defect at the SDD level is again a "Function with incorrect logic" defect located in the SDD level 1 function "generateLog". Therefore, this defect has to be investigated at the code level, as displayed in the right hand side of Fig. 19

The C function corresponding to the "generateLog" SDD level 1 function is provided. This function writes the temperature (Temp), pressure (Pres), and the three valves' positions (p1, p2 and p3, respectively) to the log file. It also reports when each valve's position is high. The code level defect report shows that the predicate "if (p3>9.6)" is erroneous, and should

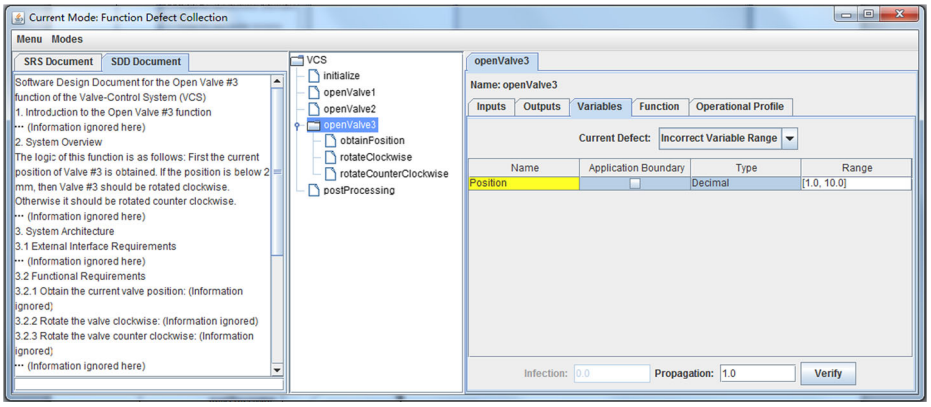


Fig. 18 Snapshot of the ARPS panel for specifying the incorrect range of "Position"

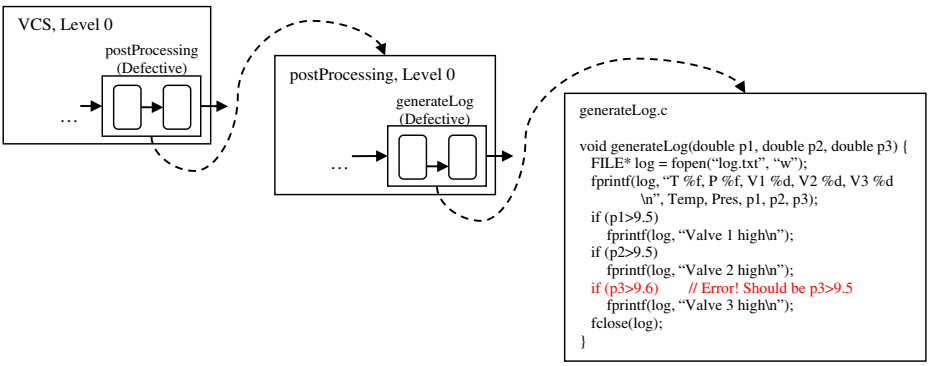


Fig. 19 How to map a SDD level defect into code level

have been written as "if (p3>9.5)". The defective portion is 9.5≤p3<9.6. This defective line of code will affect the output of the subroutine, which will be reflected in the Infection probability of the "generateLog" SDD level 1 function. Table 6 provides the information about Valve3's position, from which the probability of the defective portion of the erroneous predicate can be calculated. The rule is the same as that used for "Incorrect/ambiguous predicate" at the SRS and SDD levels:

$$\begin{aligned}\text{Prob}_{generateLog,code}\,(failure) &= E_{1,code} \times I_{1,code} \times P_{1,code} \\ &= 100\,\% \times Prob\,(9.5 \le \text{p3} < 9.6) \times 100\,\% \\ &= 100\,\% \times \int_{9.5}^{9.6} \left(\frac{2}{81} pos - \frac{2}{81}\right) dpos \times 100\,\% = 2.1\,\%\end{aligned}$$

The above failure probability at the code level is the Infection probability of the "generateLog" SDD level 1 function. The Execution and Propagation probabilities of the erroneous predicate are 100 % since the erroneous predicate is at the main logic branch and not succeeded by any other statement The failure probability of "generateLog" SDD level 1 function is hence:

$$\begin{aligned}\text{Prob}_{generateLog,SDD}\,(failure) &= E_{1,SDD} \times I_{1,SDD} \times P_{1,SDD} \\ &= 100\,\% \times 2.1\,\% \times 100\,\% = 2.1\,\% \end{aligned} \quad (9)$$

The Execution and Propagation probabilities at the SDD level are 100 % again since "generateLog" SDD level 1 function is on the main SDD logic branch and is the last function in "postProcessing". This failure probability is the Infection probability of the SRS level 1 function "postProcessing". Adding this defect into the reliability calculation in Section 8.1, the updated software reliability can be calculated. Note that the Execution probability of

"postProcessing" is 100 % since it is on the main logic branch; the Propagation probability is 100 % since this is the last function on this branch

$$\begin{aligned} Prob_{VCS}(failure) &= E_{1,SRS} \times I_{1,SRS} \times P_{1,SRS} + E_{2,SRS} \\ &\times I_{2,SRS} \times P_{2,SRS} + E_{3,SRS} \times I_{3,SRS} \times P_{3,SRS} \\ &= 85.1\,\% \times 1.2\,\% \times 10\,\% + 15.9\,\% \times 3.7\,\% \\ &\times 10\,\% + 100\,\% \times 2.1\,\% \times 100\,\% = 2.26\,\% \\ Re_{VCS} &= 1 - Prob_{VCS}(failure) = 97.74\,\% \end{aligned}$$

So far we have discussed the EFSM-based RePS methodology and the corresponding features in the ARPS tool. The authors have conducted research to compare the reliability evaluation result using the EFSM-based RePS methodology and the observed reliability of the software of a real nuclear reactor protection system (Smidts et al. 2010). The comparison results showed that the EFSM-based RePS methodology is able to provide precise evaluation results, which helps substantiate our approach.

10 Experimental Validation of ARPS' Usability

10.1 Introduction

As discussed before, one of the main reasons for developing the ARPS tool is to reduce the EFSM construction time and the error rate compared to the manual application of the EFSM-based RePS methodology. Thus an experimental validation is required to investigate whether the potential users can learn the ARPS tool and to collect their feedback as well. During our study we recruited engineering undergraduate students as our research subjects. We taught them the SRS level EFSM-based RePS methodology and the corresponding features of the tool. At the final stage of the experiment the subjects were asked to solve problems by both manually applying the methodology and using the tool. It was found that the ARPS tool not only assisted the subjects in achieving a better performance but was preferred to the manual analysis.

10.2 Research Subject Identification

From the previous sections it can be seen that the potential user of the ARPS tool should have a bachelor's degree in engineering or related disciplines. More specifically, he/she should possess the following knowledge:

1) An understanding of engineering mathematics such as calculus and probability theory.
2) An understanding of the concepts of system modeling.
3) The capability to (yet not necessarily master) program with procedural languages.

Note that although ARPS requires certain experience of programming with procedural languages, the specific requirement for this experiment is not very stringent. In addition, Huang et al. (2014) indicates that the performance of a programmer on a certain task depends on whether the programmer's knowledge base matches the expertise required by the task. If a task is new yet feasible for all, more experienced programmers do not necessarily perform better than less experienced ones. This phenomenon corresponds to the situation

encountered in the case of the ARPS tool. Therefore, it was decided to use junior/senior level engineering students from mechanical engineering, electrical engineering, computer science and engineering and integrated system engineering as research subjects. Eventually 15 subjects were recruited and were equally divided into two groups based on their year of study and major. All subjects underwent the same training sessions and test problems. However, the approaches they applied to solve the exam problems were different. I.e., when group #1 subjects were solving a certain exam problem manually, group #2 was solving the same problem using the ARPS tool, or vice versa. It will be seen that the sample size can be effectively doubled by this setting.

10.3 Research Design

The goal of this experiment is to evaluate the usability of the ARPS tool. The proposed experimental design is as follows:

10.3.1 Questions to be Answered in this Experiment.

The question to be answered in this experiment is: "Determine whether the original HLEFSM construction, the defect mapping and the reliability assessment with ARPS tool support are more usable than without tool support".

10.3.2 The Variables of the Experiment.

1) The independent variable—the software reliability modeling approach. The approach is either manual or tool-based. In other words, experiment groups either manually apply the EFSM-based RePS methodology (will be referred as ***M***) or use the ARPS tool (will be referred as ***A***).
2) The controlled variable—the background knowledge and experience of the subjects and it is measured on an ordinal scale.
3) The dependent variable—the dependent variable is the usability measures (defined below) of the software reliability modeling techniques.

10.3.3 The Usability Measures

The following six usability measures are investigated in this experiment.

1) Error Index (***EI***): ***EI*** is a score expressed as the percentage of an exam problem correctly completed by a subject. A higher ***EI*** indicates a higher performance. Table 8 provides the specific grading policy used in this study. The grading policy is based on the type of errors a subject may commit.

Each exam problem, whether solved manually or using the ARPS tool, consists of three steps: 1) reading the SRS and constructing the original HLEFSM; 2) reading the SRS level defect report and mapping the defects to the original HLEFSM; 3) conducting the Execution, Infection and Propagation analysis to assess the software reliability. The specific errors made by the subjects vary from one step to another. In step 1) and 2) the commonly made mistakes include missing links between the state transitions and an incorrect interpretation of the SRS logic; in step 3) the subjects sometimes found manually applying the analysis

Table 8 Grading policy used in our study

No	Error Type	Points obtained
1	Missing	0
2	Partially incorrect	0.5
3	Correct	1
4	Extra	−1

relating to predicates difficult. However, the grading policy in Table 8 is designed to be general for all three steps.

2) Time (***T***): ***T*** is the total amount of time to complete a test problem, in number of minutes. Each subject completed one small problem manually, one small problem using the tool, one larger problem manually and one larger problem using the tool. Therefore, which method is more time consuming can be investigated.
3) Total Number of Errors (***NE***): The total number of errors committed by a subject. This measure only counts the total number of errors committed by a subject without considering the specific error type. It is a complement of ***EI*** because the grading policy of ***EI*** is subjective, and the researcher's evaluation of the error type is subjective, too.
4) Difference between number of "missing" errors for the manual analysis and number of "missing" errors for the tool analysis (***Missing***). This measure only considers the number of "missing" errors committed by a subject. For a certain subject, this measure is defined as the number of "missing" errors committed during manual analysis minus the number of "missing" errors committed during tool analysis.
5) Difference between number of "partially incorrect" errors for the manual analysis and number of "partially incorrect" errors for the tool analysis (***Incorrect***). This measure only considers the number of "partially incorrect" errors committed by a subject. For a certain subject, this measure is defined as the number of "partially incorrect" errors committed during manual analysis minus the number of "partially incorrect" errors committed during tool analysis.
6) Ease of Learning (***Ease***): A subjective measure evaluating the ease of learning. Four levels of ease are considered: 4-very easy, 3-easy, 2-moderately difficult, 1-very difficult. The subjects were asked to select one of the four options.
7) Satisfaction (***Sat***): A subjective measure evaluating the degree of subject satisfaction. Four levels of satisfaction are considered: 4-very satisfied, 3-satisfied, 2-moderately dissatisfied, 1-very dissatisfied. The subjects were asked to select one of the four options.

10.3.4 Hypothesis Test

The general null hypothesis (H_0) is that there is no difference between the usability measures of the manual analysis (***M***) and those of the ARPS tool analysis (***A***). The general alternative hypothesis (H_A) is that the difference in the usability measures of ***M*** and ***A*** is significant. These two hypotheses are tailored for different measures. Based on the normality of the data, different statistical testing methods are used.

Table 9 The design of experiment

		Testing Session #1	Testing Session #2		
Group1	Training	MT1	AT2	MT3	AT4
Group2	Training	AT1	MT2	AT3	MT4

10.3.5 The Design of Experiment

Table 9 summarizes the design of experiment. Group #1 and Group #2 subjects were lectured using the same materials and instructors. However, they did not know that they would be divided into two groups and taking the test in the way shown in Table 9. This is helpful to avoid the potential bias caused by grouping the subjects. The training session lasted three and half days. The two test sessions amount to 1.5 days. The duration for Session #1 and Session #2 was half day and one day, respectively. There was no time limit for each testing session, and the subjects spent approximately 3 hours for Session #1 and 6 hours for Session #2. There were two exam problems in both sessions. In Table 9 MT1 stands for "manually solve Test #1" and AT2 stands for "solve Test #2 using the ARPS tool", respectively. The size of Test #1 and #2 are identical and similarly the size of Test #3 and #4 are identical. The size is determined by the number of state transitions and predicates in the HLEFSM. The size of Test #1 and #2 are approximately 10 units and the size of Test #3 and #4 are approximately 50 units. in scale is a factor of 5.

The main advantage of this design of experiment is that the data collected from Test #1 and Test #2 (i.e., the two small scale problems) can be combined, and the data collected from Test #3 and Test #4 (i.e., the two larger scale problems) can be combined. Therefore, the sample size effectively doubles. Eventually 15 data points were obtained for both the small scale problems and the larger scale problems.

10.4 Training

The training session lasted 3.5 days. There were two 80 minutes classes each morning and each afternoon. Between the two classes there was a 15 minutes break. The training session was conducted in a typical classroom and the materials were displayed on slides. Because

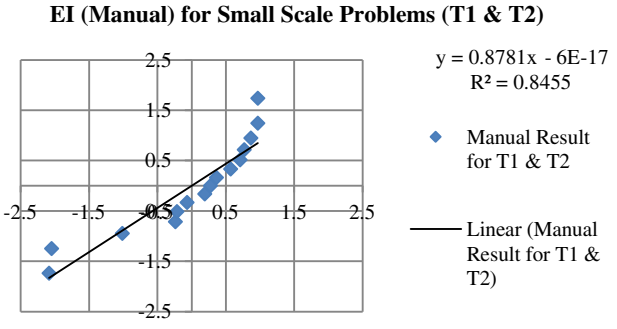


Fig. 20 Normal probability plot for the manual analysis of the small scale problems

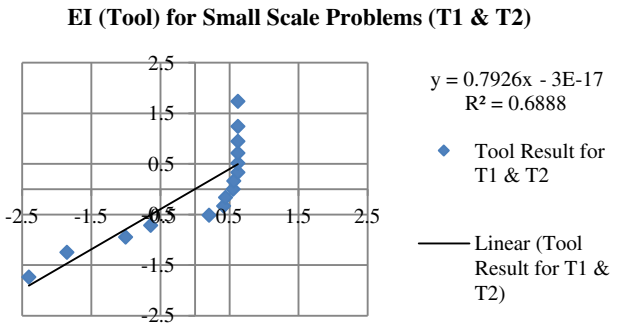


Fig. 21 Normal probability plot for the tool analysis of the small scale problems

the training session was fairly short, only a portion of the methodology and corresponding tool features were discussed. More specifically, the lectures were about the SRS level, the Execution and Infection analysis for the following four defects: 1) Missing Instance of Function; 2) Function with Incorrect Logic; 3) Missing Predicate and 4) Incorrect/Ambiguous Predicate. The Propagation analysis was not introduced because of the time limit. All Propagation probabilities were directly provided to the subjects. No homework assignments were given but the students were encouraged to review the class materials by themselves.

10.5 Data Analysis

10.5.1 Error Index (EI)

Figures 20 and 21 display the normal probability plots (Chambers et al. 1983) for the manual analysis and the tool analysis for the small scale problems (i.e., Test #1 and #2), respectively. In both cases the data is not normally distributed. The results obtained for manual and tool for Test #3 and #4 are not normally distributed, either (but they were not shown due to space limitation). Therefore, the traditional t-test for difference is not applicable and nonparametric testing methodologies are considered. Both Wilcoxon test (Siegel 1956) and Sign test

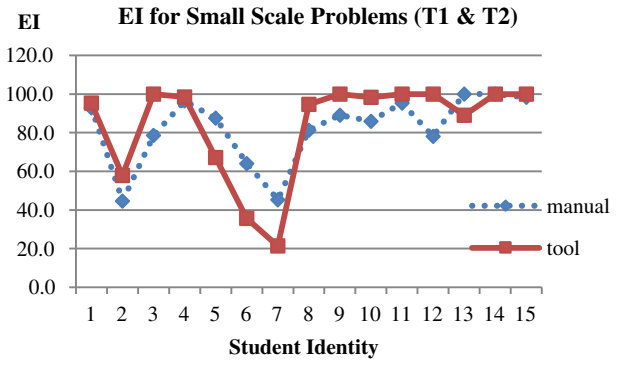


Fig. 22 *EI* of the small scale problems

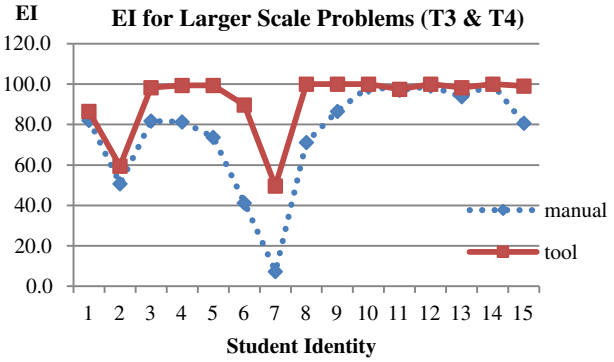


Fig. 23 *EI* of the larger scale problems

(Mendenhall et al. 1989; Dixon 1953) are commonly used nonparametric methodologies. However, the Wilcoxon test requires a symmetric distribution of the data, a requirement which is not satisfied in this case. Therefore, the Sign test was applied.

Figures 22 and 23 provide plots for ***EI*** for the small scale problems and the larger scale problems, respectively. For the small scale problems, the tool result is not obviously superior to (higher than) the manual result, since the two lines cross each other. However, for the larger scale problems, the tool result is systematically superior to the manual result. Sign test was applied to both the small scale problems and the larger scale problems. The statistics are summarized in Table 10.

From Table 10 it can be seen that the two-tailed p-value for the small scale problems is fairly large (0.180), which means that the null hypothesis cannot be rejected at the significance level of $\alpha = 0.05$, i.e., the tool ***EI*** is not obviously different from the manual ***EI***. The statistical power is also small (0.195). However, for the larger scale problems, the p-value is very small (1E-4), which means that the tool ***EI*** is very different from the manual ***EI***. The corresponding statistical power is large (0.970). The 98.7 % confidence intervals are also listed in the table (98.7 % is calculated based on the total number of non-zero data points, which is 14). Similar inference can be obtained: for the small scale problems the tool ***EI*** is not obviously better than the manual ***EI***, since the confidence interval is symmetric

Table 10 EI statistics summary (a score expressed as the percentage of a task correctly completed by a subject)

	$EI_{\text{Tool}} - EI_{\text{Manual}}$ (%)	
Tests	Test 1&2	Test 3&4
No. data points	15 (14 nonzero)	15 (14 nonzero)
Mean	1.4	15.5
Standard Deviation	15.8	15.3
p-value (two-tailed)	0.180	1E-4
Statistical Power ($\alpha \leq 0.05$)	0.195	0.970
Confidence Interval (CI)	98.7 % CI is (-20.3, 13.4)	98.7 % CI is (1.0, 28.9)

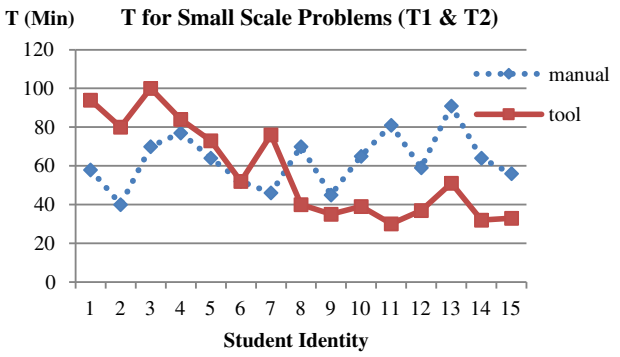


Fig. 24 *T* for the small scale problems

with respect to 0. However, for the larger scale problems the tool ***EI*** is seemingly superior to the manual ***EI***, since the entire confidence interval is above 0.

10.5.2 Time (T)

The measure ***Time*** does not follow a normal distribution either. Hence the Sign test is used. (In fact none of the measures follow a normal distribution and hence the Sign test is used in all cases). Figures 24 and 25 provide the plots for ***T*** for the small scale problems and larger scale problems, respectively. For the small scale problems the manual analysis and the tool analysis prove to be equally time consuming. However, for the larger scale problems the tool analysis saves time for 12 out of 15 subjects. Table 11 lists the specific statistics: for the small scale problems the p-value is large (0.791) and the statistical power is small; however, for the larger scale problems the p-value is small (0.035), meaning that the difference between the manual ***T*** value and the tool ***T*** value is significant. The statistical power (0.648) is also much larger than that observed for small scale problems.

10.5.3 Total Number of Errors (NE)

Figures 26 and 27 display the plots for ***NE*** and Table 12 lists the specific statistics.

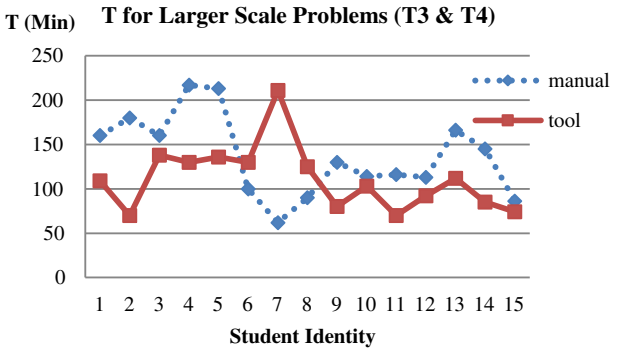


Fig. 25 *T* for the larger scale problems

Table 11 T statistics summary

Tests	$T_{\text{Manual}} - T_{\text{Tool}}$ (Min) Test 1&2	Test 3&4
No. data points	15 (14 nonzero)	15 (15 nonzero)
Mean	5.5	25.8
Standard Deviation	29.6	62.6
p-value (two-tailed)	0.791	0.035
Statistical Power ($\alpha \leq 0.05$)	0.028	0.648
Confidence Interval (CI)	94.3 % CI is (-30.0, 30.0)	96.5 % CI is (11.0, 60.0)

Figure 26 displays that the manual **NE** for small scale problems crosses the tool **NE**, which indicates that the difference is not significant. The p-value is 0.180, which is fairly large. The 98.7 % confidence interval is (-4, 6) which contains 0. However, for the larger scale problems the difference is significant as illustrated in Table 12. The statistics in Fig. 27 shows the same conclusion. The p-value is very small (0.002) and the 98.7 % confidence interval is above 0. The power of the test is also large (0.915). Therefore, it can be concluded that the tool ***NE*** for the larger scale problems is superior to (i.e., smaller than) the manual ***NE***. The subjects tend to make fewer mistakes with the assistance of the tool.

10.5.4 Number of "Missing" Errors (Missing) and "Partially Incorrect" Errors (Incorrect)

The tool is effect on the number of "missing" errors and "partially incorrect" errors is also investigated. Note that the number of "extra" errors is not considered because there are too few to support any statistically meaningful comparisons.

Figure 28 displays the plot for (the number of errors of type) ***Missing***, where the difference between the manual ***Missing*** and the tool ***Missing*** is directly shown. When the test problems are small, the tool does not really help reduce the number of "missing" errors, since the plot oscillates around the horizontal axis. However, when the problems become larger, the manual ***Missing*** is much larger than the tool ***Missing***, implying that the tool significantly reduces the number of "missing" errors. Table 13 provides the detailed statistical

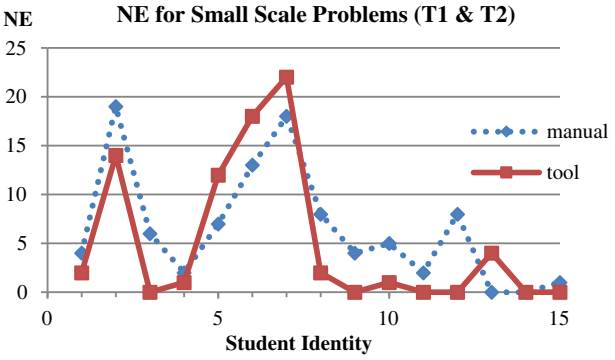


Fig. 26 NE for small scale problems

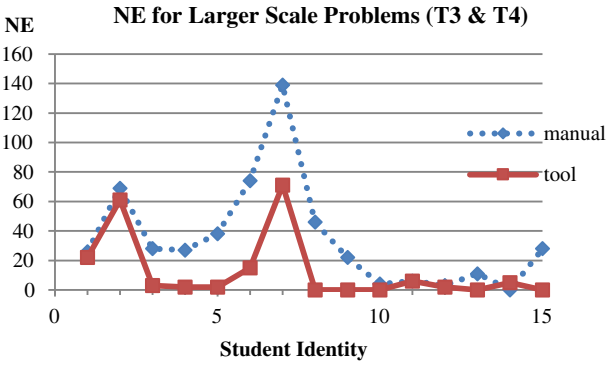


Fig. 27 NE for larger scale problems

results: For the small scale problems the p-value is very large (0.774) meaning that the null hypothesis cannot be rejected, while for the larger scale problems the p-value is very small (0.006). Both confidence intervals support the statistical inference, too.

Figure 29 provides the plot for (the number of errors of type) ***Incorrect***, where the difference between the manual ***Incorrect*** and the tool ***Incorrect*** is displayed. In this case both plots are mostly above the horizontal axis, which means that the tool is useful for mitigating "partially incorrect" errors, regardless of the problem scale. Table 14 shows that both p-values are small. However, it should be noted that for both cases the statistical power is medium.

10.5.5 Ease of Learning (Ease)

Figure 30 displays the plot for ***Ease***, which is designed to investigate the subjects' perception of the ease of use of the two methodologies. For all subjects the tool ***Ease*** is higher than or equal to the manual ***Ease***. However, in only 4 instances is the difference different from zero and hence the statistics are not adequate: The two-tailed p-value is 0.125, which is not small enough to make any meaningful inference. Neither confidence interval nor statistical power can be calculated because of the small number of non-zero data points.

Table 12 T statistics summary

	$NE_{\text{Manual}} - NE_{\text{Tool}}$	
Tests	Test 1&2	Test 3&4
No. data points	15 (14 nonzero)	15 (14 nonzero)
Mean	1.4	22.1
Standard Deviation	4.3	22.2
p-value (two-tailed)	0.180	0.002
Statistical Power ($\alpha \leq 0.05$)	0.195	0.915
Confidence Interval (CI)	98.7 % CI is (-4, 6)	98.7 % CI is (1, 46)

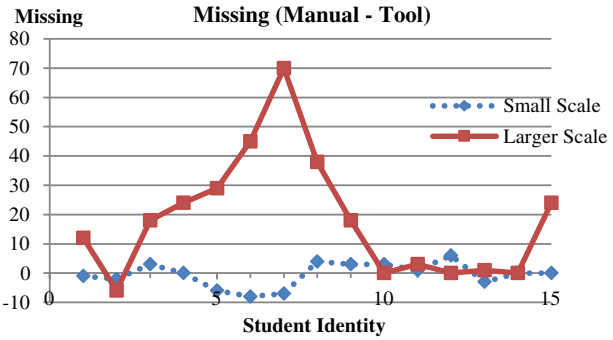


Fig. 28 Plot for Missing

10.5.6 Satisfaction (Sat)

Figure 31 displays the plot for ***Sat***, which is designed to investigate the subjects' perceived level of satisfaction with the two methodologies. For 14 out of 15 subjects, the level of satisfaction with the tool is equal to or higher than the level of satisfaction with the manual analysis. However, because the number of non-zeros is small (6), the statistics are inadequate: the two-tailed p-value is as high as 0.219 and the statistical power is still fairly low (0.339).

10.5.7 Summary of all Results

Table 15 summarizes all results. As already discussed, the usability measures for the ARPS tool are not superior to those for the manual analysis when applied to the small scale problems (Test 1&2): for ***EI***, ***T***, ***NE*** and ***Missing***, the two-tailed p-values are large and the statistical powers are small. However, for the larger scale problems (Test 3&4) these four usability measures indicate that the tool not only helps mitigate various types of errors but also saves time for the analyst. All four p-values are small and the corresponding statistical powers are fairly large. The measure ***Incorrect*** is slightly different from the others: for both small scale and larger scale problems the p-values are small and the statistical powers are medium. While there is evidence that the tool helps avoid "Partially incorrect" errors, the statistics indicate that further evidence is needed.

Table 13 NE statistics summary

	Missing	
Tests	Test 1&2	Test 3&4
No. data points	15 (12 nonzero)	15 (12 nonzero)
Mean	-0.5	18.4
Standard Deviation	4.1	20.9
p-value (two-tailed)	1.000	0.006
Statistical Power ($\alpha \leq 0.05$)	0.039	0.920
Confidence Interval (CI)	96.1 % CI is (-6, 3)	96.1 % CI is (3, 38)

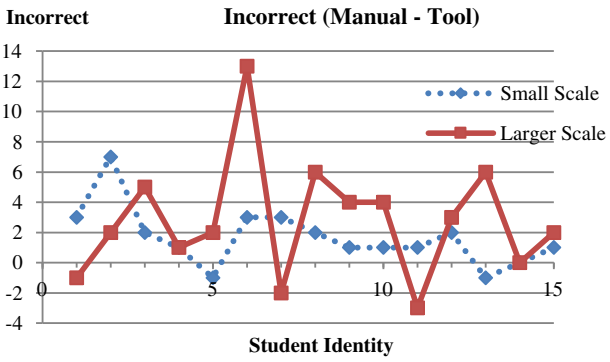


Fig. 29 Plot for Incorrect

Observation of subjects' performance shows that the automation features of the ARPS tool help avoid errors. When the subjects applied the defect templates to map the defects manually, they made mistakes in drawing the modified branches. This did not occur during the tool analysis since the tool automatically generated the modified branches. Mistakes in the modified HLEFSM also propagated to the reliability assessment, which further deteriorated the manual results. Additional mistakes were made during the manual Execution, Infection and Propagation analysis, which can be effectively avoided by using the tool as well. Therefore, automation features are the most valuable contribution of our tool and hence more such features should be introduced in the future.

10.5.8 Threats to Validity

Internal validity The subjects used in this experiment are different from the potential users of the ARPS tool, which may pose threats to validity. In this subsection possible threats to internal validity are discussed.

1) Selection bias

Our research subjects are junior and senior engineering undergraduate students. Although it is possible that senior students are more knowledgeable than juniors, the difference should not affect the results. The reason is that the EFSM-based RePS methodology

Table 14 *Incorrect* statistics summary

	Incorrect	
Tests	Test 1&2	Test 3&4
No. data points	15 (14 nonzero)	15 (14 nonzero)
Mean	1.7	2.8
Standard Deviation	2.0	3.9
p-value (two-tailed)	0.013	0.057
Statistical Power ($\alpha \leq 0.05$)	0.676	0.4
Confidence Interval (CI)	94.3 % CI is (1, 3)	94.3 % CI is (1, 5)

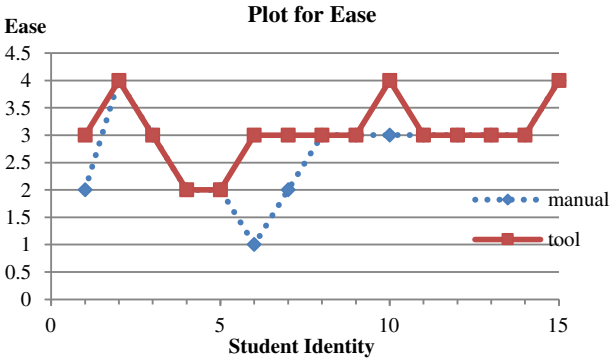


Fig. 30 Plot for Ease

and ARPS tool are new to all subjects. They are not based on pre-existing knowledge selectively accessible to some of the subjects. In addition, the difference in subjects' major should have no effect on the results. This is because learning our techniques requires an average level of programming knowledge and mathematics shared by all engineering students.

2) Rivalry

All subjects were trained identically and did not know a priori that they would be grouped. Thus there was no possible rivalry between the two groups during the training session. The subjects were informed that the exams taken by each group were the same. However, they did not know that the techniques applied by the other group to solve the same problems were actually different. Therefore, the desire to out-perform the other group should be minimized.

3) History

The duration of the entire experiment was only 5 days and it was during the summer break. Thus the possibility that any events outside the experiment happened to all 15 subjects and further caused changes to their attributes can be ignored.

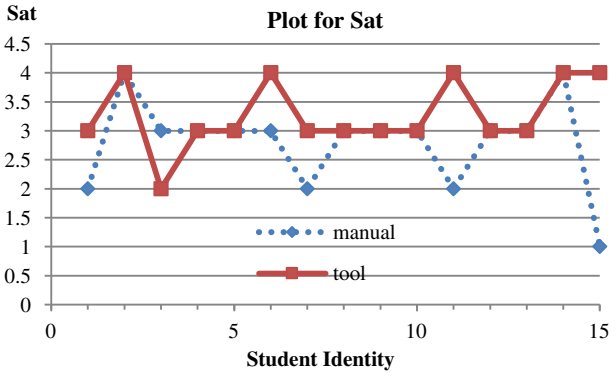


Fig. 31 Plot for Sat

Table 15 Summary of all the results

	Small Scale			Larger Scale		
Measure	Two-tailed p-value	Power	Hypothesis Accepted?	Two-tailed p-value	Power	Hypothesis Accepted?
$EI_{\mathbf{Tool-Manual}}$	0.180	0.195	No	1E-4	0.97	Yes
$T_{\mathbf{Manual-Tool}}$	0.791	0.028	No	0.035	0.648	Yes
$NE_{\mathbf{Manual-Tool}}$	0.180	0.195	No	0.002	0.915	Yes
$Missing_{\mathbf{Manual-Tool}}$	1.000	0.039	No	0.006	0.920	Yes
$Incorrect_{\mathbf{Manual-Tool}}$	0.013	0.676	Yes	0.057	0.4	No
Ease	Based on observation, tool is equally or easier to learn than manual analysis					
	P-value is 0.125. Power cannot be calculated					
Sat	Based on observation, tool is equally or more satisfying than manual					
	analysis. P-value is 0.219. Power is 0.339					

4) Maturation in skill level

Because the duration of the training was only 3.5 days and the subjects were only encouraged to review the materials, it is possible that their level of skill improved during the exams. However, because (1) two small tests were completed by the subjects in the first test session where no time limit was assigned and (2) these two tests were sufficiently complex, it is believed that the subjects reached sufficient level of maturation. In other words, even if the results for the small scale tests may suffer a loss of internal validity due to maturation, the results for the larger scale tests should not.

5) Repeated testing

Although exam problems with the same format were used to test the subjects four times, the possible gain in score does not affect the result. First, the scale of the two problems in the second day of tests is 5 times larger than the one used in the first day of tests. This helps mitigate the gain in score. Second, the phenomenon of interest is the difference between manual and tool analysis. Even if the subjects did better in the second day of tests, as long as the difference between the subjects' skill for manual and tool analysis is preserved, the experimental result is not affected.

6) Mortality/differential attrition

Two subjects dropped after the first morning session. However, only background information and an introduction to the techniques was discussed during that session. Thus this bias does not exist.

External validity Our research subjects are junior and senior engineering undergraduates. This is because the average users of our tool are from engineering fields with at least a bachelor's degree. Therefore, generalizing our findings should be valid since our subjects have similar background and learning potential as the expected users. Although it is possible that the expected users are more knowledgeable than the subjects in their particular field of expertise, learning our methodologies just requires an average level of proficiency in mathematics, programming and modeling. These characteristics are shared by the potential

users and the student subjects. In addition, it is expected that both students and industry users are equally novices with regard to this particular technology.

The four tests used during the experiment were designed in such a way that all aspects of the techniques available at that time were covered. In addition, the example systems used in the tests are independent of the techniques. Different scales were considered as well. Therefore, possible biases introduced by the tests themselves should be minimized, and should not affect external validity.

11 Conclusions and Future Research

In this paper the EFSM-based RePS methodology, the ARPS tool and the experimental validation are discussed. The ARPS tool extends and implements the EFSM-based RePS methodology, which is developed to model and calculate software reliability for safety related software. The EFSM-based RePS methodology is based on the software development artifacts and the operational profile information. The SRS and SDD are used to develop the system representation, which is based on EFSM. Defect templates are introduced for commonly found software defects, which are used to map the defects to the EFSM representation. The Execution, Infection and Propagation analysis are developed for software reliability calculation.

An experiment which uses human subjects was conducted to evaluate the usability of the ARPS tool. During this experiment 15 engineering undergraduate students were recruited, trained and tested. The result shows that the ARPS tool can help the subjects avoid mistakes during their analysis as well as reduce the criticality of the mistakes. The subjects seemed to display a higher level of satisfaction when using the tool and felt the tool was easier to learn.

The future research will be three-folds. First, the theory for the Propagation analysis should be completed and implemented into the ARPS tool. Second, the code level analysis should be developed and added into the tool, and the combination and connection between different hierarchies of the tool need to be studied. Third, the experimental validation on the other measures of the ARPS tool should be conducted, with more subjects recruited, more features of the tool involved and more complex exam problems used.

Acknowledgments This paper was prepared as an account of work sponsored by an agency of the U.S. Government. Neither the U.S. Government nor any agency thereof, nor any of their employees, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for any third party's use, or the results of such use, of any information, apparatus, product, or process disclosed in this report, or represents that its use by such third party would not infringe privately owned rights. The views expressed in this paper are not necessarily those of the U.S. Nuclear Regulatory Commission. We are grateful to Kevin Smearsoll and Boyuan Li for supporting this research.

Appendix

Appendix. List of Acronyms Used in this Paper

A	Solve a test problem using ARPS tool
ARPS	Automated Reliability Prediction System
BBN	Bayesian Belief Networks
EFSM	Extended Finite State Machine

E_i	The Execution probability of the i-th defect
EI	Error index
F	False
H	Null hypothesis
H_A	Alternative hypothesis
HLEFSM	High Level Extended Finite State Machine
IAP	Incorrect/Ambiguous Predicate
I_i	The Infection probability of the i-th defect
INP	Information Not Post-processed
IP	Information Post-processed
IV	Internal Variables
LLEFSM	Low Level Extended Finite State Machine
M	Manually solve a test problem
NE	Number of errors
OP	Operational Profile
P	Set of Predicates; Pressure
P_C	The correct predicate
pdf	Probability density function
P_i	The Propagation probability of the i-th defect
PIE	Propagation, Infection and Execution analysis
pos	Position
P_O	The original predicate
Pres	Pressure
prev	Previous
Prob	Probability
Re	Reliability
RePS	Reliability Prediction System
S	Set of States
Sat	Satisfactory
SDD	Software Design Document
SI	State Initialized
SNI	State Not Initialized
SRGM	Software Reliability Growth Model
SRS	Software Requirement Specification
SUS's	Software under study's
T	Set of Transactions; Temperature; True
T	Time
T1	Test #1
T2	Test #2
Temp	Temperature
V1_C	Valve #1 closed
V1_O	Valve #1 opened
V2_C	Valve #2 closed
V2_O	Valve #2 opened
V3_C	Valve #3 closed
V3_O	Valve #3 opened
VCS	Valve Control System
Γ	Output Variables
Σ	Input Variables

References

IEEE (1990) IEEE Standard Glossary of Software Engineering Terminology, IEEE Std.610.12-1990. IEEE, New York

ISO/IEC (2001) ISO/IEC 9126-1: 2001, Software Engineering – Product Quality – Part 1: Quality model

Musa J (1975) A theory of software reliability and its application. IEEE Trans Softw Eng 1(3):312–327

Huang C (2005) Performance analysis of software reliability growth models with testing-effort and change-point. J Syst Softw 76(2):181–194

Huang C, Kuo S et al. (2007) An assessment of testing-effort dependent software reliability growth models. IEEE Trans Reliab 56(2):198–211

Mills H (1972) On the statistical validation of computer programs. IBM Federal Systems Division Report:72–6015

Walia G, Carver J (2008) The Effect of the Number of Defects on Estimates Produced by Capture-Recapture Models. In: Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on, pp. 305-306

Li M, Smidts C (2003) A ranking of software engineering measures based on expert opinion, vol 29, pp 24–811

Pham H (2007) System software reliability. Springer

Smidts C, Huang F et al. (2015) A Method for Quantifying the Dependability Attributes of Software-Based Safety Critical Instrumentation Control Systems in Nuclear Power Plants. In: Proc. NPIC-HMIT 2015

Huang F, Liu B (2013) Study on the correlations between program metrics and defect rate by a controlled experiment. Int J Softw Eng 7(3):114–120

Huang F, Liu B et al. (2015) The impact of software process consistency on residual defects. Journal of Software Evolution and Process

Smidts C, Li M (2004) Validation of A Methodology for Assessing Software Quality. NUREG/CR-6848, Office of Nuclear Regulatory Research, Washington DC

Smidts C, Li M et al. (2010) A Large Scale Validation of a Methodology for Assessing Software Reliability. NUREG/CR-7042, Office of Nuclear Regulatory Research, Washington DC

Li X, Gupta J et al. (2013) ARPS: An Automated Reliability Prediction System Tool for Safety Critical Software, PSA 2013, Columbia, South Carolina, September 22-27

Wang CJ, Liu MT (1993) Generating Test Cases for EFSM with Given Fault Models. In: Proceedings of 12th IEEE Computer and Communications Societies

Voas J (1992) PIE: A Dynamic Failure-Based Technique. IEEE Trans Softw Eng 18(8)

Lyu M (1996) Handbook of software reliability engineering. Vol. 222. IEEE computer society press, CA

Smidts C, Li B et al. (2002) Software Reliability Models, vol 2, 2nd ed. Wiley, New York, pp 1594–1610

Pandey A, Goyal N (2013) Early Software Reliability Prediction. Springer

Cheung L, Roshandel R et al. (2008) Early prediction of software component reliability. In: Proceedings of the 30th international conference on Software engineering, pp. 111–120

Gaffney G, Pietrolewiez J (1990) An automated model for software early error prediction (SWEEP). In: Proceeding of 13th Minnow Brook Workshop on Software Reliability

Fenton N, Neil M (1999) A critique of software defect prediction models. IEEE Trans Softw Eng 25(5):675–689

Langseth H, Portinale L (2007) Bayesian networks in reliability. Reliab Eng Syst Saf 92(1):92–108

Gokhale S, Trivedi K (2006) Analytical models for architecture-based software reliability prediction: a unification framework. IEEE Trans Reliab 55(4):578–590

Lyu M, Nikora A (1992) CASRE: a computer-aided software reliability estimation tool. In: Computer-Aided Software Engineering, 1992. Proceedings., Fifth International Workshop on, pp. 264-275

Ramani S, Gokhale S et al. (2000) SREPT: software reliability estimation and prediction tool. Perform Eval 39(1):37–60

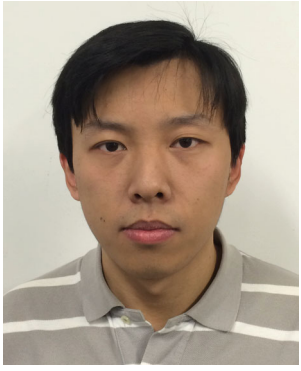
Chen C, Lin C et al. (2006) CARATS: a computer-aided reliability assessment tool for software based on object-oriented design. In: TENCON 2006. 2006 IEEE Region 10 Conference, pp. 1-4

Wang W, Scannell D (2005) An architecture-based software reliability modeling tool and its support for teaching. In: Frontiers in Education, 2005. FIE'05. Proceedings 35th Annual Conference, pp. T4C-T4C

Boudali H, Dugan J (2006) A continuous-time Bayesian network reliability modeling, and analysis framework. IEEE Trans Reliab 55(1):86–97

IEEE Computer Society (1998) Software Engineering Standards Committee, and IEEE-SA Standards Board. IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 830 -1998, Institute of Electrical and Electronics Engineers

Musa J, Lannino A et al. (1987) Software Reliability-Measurement, Prediction, Applications. McGraw-Hill, New York
Musa J (1993) Operational profiles in software-reliability engineering. Software, IEEE 10(2):14–32
Lam M, Sethi R et al. (2006) Compilers: Principles, Techniques, and Tools
Wolfram (2014) Equation solving. http://reference.wolfram.com/language/guide/EquationSolving.html. [Retrieved: 2014-10-14]
Mathworks (2014) Solve equations and inequalities. http://www.mathworks.com/help/symbolic/mupad_ref/solve.html. [Retrieved: 2014-10-14]
IEEE Computer Society (1998) Software & System Engineering Standards Committee, IEEE Standard for Information Technology—Systems Design—Software Design Descriptions IEEE Std 1016-1998, Institute of Electrical and Electronics Engineers
Booch G, Rumbaugh J et al. (2005) Unified Modeling Language User Guide, the 2nd Edition. Addison-Wesley
Huang F, Liu B et al. (2014) The links between human error diversity and software diversity: Implications for fault diversity seeking. Science of Computer Programming 89(Part C):350–373
Chambers J, Cleveland W et al. (1983) Graphical Methods for Data Analysis. Wadsworth
Siegel S (1956) Non-parametric statistics for the behavioral sciences, New York: McGraw-Hill, pp 75–83
Mendenhall W, Wackerly D et al. (1989) 15: Nonparametric statistics, Fourth ed. PWS-Kent, pp 674–679
Dixon W (1953) Power functions of the sign test and power efficiency for normal alternatives. Ann Math Stat:467–473
Li B, Li M et al. (2005) Integrating software into PRA: A software-related failure mode taxonomy. Risk Anal 26(4)



Dr. Xiang Li received his PhD from the Nuclear Engineering Program in the Ohio State University (OSU) in 2015. Before joining to OSU he obtained his B.S and M.S from the Department of Engineering Physics in Tsinghua University. His research interests include software reliability, automatic software testing and software safety. His research has been funded by government agencies such as FAA, NRC, and DOE.

Dr. Chetan Mutha has a Phd in Mechanical Engineering from The Ohio State University advised by Professor Smidts. His research interests include systems and software reliability assessment, integrated system design and analysis, and fault diagnosis early in the design phase. He has published several conference and journal papers. His research has been funded through government agencies such as the Air Force Office of Scientific Research (AFOSR), DoD, and NRC. Currently, he works as technical advisor at Oblon, llc.

Dr. Carol S. Smidts is a Professor in the Department of Mechanical and Aerospace Engineering at Ohio State University. She graduated with a BS/MS and PhD from the Université Libre de Bruxelles, Belgium, in 1986 and 1991, respectively. She was a Professor at the University of Maryland at College Park in the Reliability Engineering Program from 1994 to 2008. Her research interests are in software reliability, SW safety, SW testing, PRA, and human reliability. She is a senior member of the Institute of Electrical and Electronic Engineers; an Associate Editor of IEEE Transactions on Reliability; and a member of the editorial board of Software Testing, Verification, and Reliability.