CHAPTER 6

CONCLUSIONS & FUTURE SCOPE

Software reliability predictions are leveraged by the proposed framework to find an optimal software configuration with respect to reliability. However, the science of software reliability suffers from many challenges.

6.1 CONCLUSIONS

All software reliability estimation models calculate software reliability using some statistical function. These distributions utilize certain parameters whose values are estimated based on certain assumptions using existing failure data sets. However, inaccuracy of these model assumptions in comparison to real software execution is questionable [1]. Further failure data sets are largely obtained during testing or earlier phases of software development and they may not depict actual software usage.

We have postulated that it is possible to overcome all software reliability issues by providing a mechanism to obtain runtime software representation to model the probability of next state transition in software. The major contribution of this work is a reliability prediction technique that takes into account runtime nature of software and user input to control continued software service operation.

The work we present further enhances the software reliability science with the following other notable contributions:

- The model is an attempt to confirm reliability modelling to a standard in order to establish a generic solution for software reliability engineering that can be applied to varying software systems at runtime by the software itself.
The model is based on the fact that software during execution is an automaton. Hence, we hypothesize that reliability of software at runtime is analogous to the reliability of its automata.

Through executable software code we synthesize a deterministic probabilistic behaviour model for each component module of the software under study. This model is then extended to model the probabilistic occurrence of next state transition. The resulting probabilistic component model can then be composed in parallel to estimate the overall reliability of the component-based software according to Cheung’s user-oriented reliability model [2].

The model utilizes actual assembly Opcodes generated by executable software code for ensuring fault-free software operation.

The model does not use any faulty assumptions related to the number of faults in software or process of debugging. Instead the model uses runtime state to state transition information to track the software path of execution.

The model does not require any kind of test data to generate software reliability estimates. Instead the model makes software as well as each node reliability estimate from the actual system usage information.

The novelty of the model is that it does not use post failure observations (or data) with dubious statistical distributions. Rather; it obtains all required estimates during software execution (runtime state to state transition).

The model tracks software execution path and can also be used to estimate path reliability.

Theory of software reliability has been retrofitted to software [1-2]. Many different hardware reliability models are also being applied to software (ex: Power Model, Crow, 1974). The proposed automata-based software reliability model is unique as it is based on the true nature of software execution.

The model does not try to predict or assume the nature of processes applied to debug software (imperfect/perfect debugging) [1]. Instead the model spends effort on analyzing, diagnosing, correcting and documenting software path of execution at runtime.
• Conventional models attempt to estimate reliability through testing \cite{2-3}. However, the accuracy of the estimates is governed by the thoroughness of test coverage. Automata-Based Software reliability model is not dependent on the testing process which may vary significantly from the actual system usage and may not be done exhaustively.

• Our automata-based approach for reliability prediction introduces new attributes into reliability model, i.e. the probability of transition between states and component reliability.

6.2 LIMITATIONS OF THE PROPOSED MODEL

Unlike its conventional counterparts the automata-based reliability model is an extension to the concept of effective path selection \cite{2-3} from a control flow graph representing software. However, the model does not attempt to estimate all possible paths in automata. This is impossible for software systems as software-based automata representations may have loops.

Further though the model has the capacity to identify all software faults or errors in the program and can be implemented in the execution engine to enable continuous software operation. However, as software and hardware work together in computers, the model shall be unable in preventing failure due to hardware problems. To overcome this limitation, the model will have to be used along with certain hardware fault tolerance techniques, where a faulty hardware resource can be replaced immediately without halting software operation.

Use of finite state representation technology suffers from the state explosion problem \cite{5}. The size of a finite state model may increase exponentially as the number of components grows. However, automata-based reliability model controls this problem to a large extent in the following way:

The model generates a unique state for a unique opcode instruction. The instruction set for every programming language is finite. Hence the next_state transition table for each runtime software code shall be represented through finite state model irrespective of code size.
Further though the model has the capacity to identify all software faults or errors in the program and can be implemented in the execution engine to enable continuous software operation. However, as software and hardware work together in computers, the model shall be unable in preventing failure due to hardware problems. To overcome this limitation, the model will have to be used alongwith certain hardware fault tolerance techniques, where a faulty hardware resource can be replaced immediately without halting software operation. Thus the model in combination with suitable hardware fault tolerance techniques can be extended to a complete system reliability control model.

6.3 SUGGESTIONS FOR FUTURE WORK

An approach to software reliability based on theory of automata has been derived. Various methods based on the theory of probability and statistics have been used to assess reliability of elements in hardware structures \(^{[4-5]}\). This approach establishes a model that is simple, mathematically verifiable and directly implementable on software at runtime. The model permits the monitor and control of software execution at runtime. The state transition information collected on basis of opcode constituting program code can then be linked to actual state transition of the software at runtime. In case the software acquires an allowable transition, then it is performing reliably. However any non-allowable transition indicates that the software shall execute a fault. The model can work as a control tool on top of any software to avoid fault execution. Further if we integrate this model directly at the intermediate code generation phase of a compiler, engineers can decide about the feasibility of a software project.

Our conclusion from this work is rather positive. All conventional reliability models give unacceptably optimistic reliability predictions. The proposed model does not attempt at any kind of data-driven predictions. Instead, the model controls reliability at runtime through the use of a next_state transition knowledgebase. The next_state transition knowledge base uses opcode instructions to calculate the next software state. Preliminary testing of the model has been done. However, its establishment as a generic model requires its
conversion to a software tool. We are working on the same at the time of this writing. Though the proposed model is successful in controlling and avoiding all software faults. However, the approach shall be unable to control hardware faults that may result in software failure.

This thesis provides a framework for the control of software reliability using runtime software representations. The significant contributions of this work are as follows:

- The use of opcode instructions as the transitions to the next software state.
- The algorithm to monitor software path of execution and force software halt before executing a fault.
- The application of PFSM to program analysis

I believe that I have provided strong evidence to support my thesis, which was:

Software executes as an automaton, hence the reliability of the software can be controlled by controlling correct execution of the automata.

The good news for the software reliability community with this thesis is that one can control software reliability by representing runtime program code as automata. Using this approach early decision can be made on feasibility of software projects and coding practices. Hence, decisions on continuing or discarding software can be made early in software life. Thus saving time, money and labor.

We have laid out what is required for the implementation of the proposed framework into a utility tool. The discussed challenges and problems are theoretical limitations that can be overcome and the framework can be realized.

- The automata-based approach with its strong mathematical foundation in the theory of automata is an ideal choice for a generic, intelligent, self-learning reliability model for complex, ever-evolving software systems.
- The automata-based model can be extended to ensure maximum software reliability by halting the software from transiting to an error node. From here the
software can decide whether to invoke healing or not and thus handle failure conditions in real-time.

• The model can be realized as a state-space system which can be further supported with problem solving processes for embedding intelligence and self-learning capabilities in the software itself.

• The capabilities of the automata-based software reliability model can prove beneficial in reliability estimation and forecasting of complex, ever-evolving software systems in real-time.

• The model can form the basis for inculcating self-healing properties in software system such that whenever a system arrives at error state, the system should be capable enough to recall its previous correct state and retreat back to the same in order to resume back its operation.
REFERENCES


