Software Testing for Embedded Systems

Dinesh Kumar Saini
Associate Professor, Faculty of Computing and Information Technology, Sohar University, Sohar, Oman
Research Fellow, Faculty of Engineering and Information Technology, University of Queensland, Australia

ABSTRACT

In the recent years, embedded systems have become so complex that the development and testing time is becoming extremely time consuming. As embedded systems include more and more functions for new services, embedded systems are presenting challenges with respect to the attributes of security, scalability availability, and performance with deterministic behavior. This paper presents the issues that affect testing process and technologies, which can be ameliorated by Rational Test Real-Time (RT-RT). Generally Object Oriented Approach is adopted while designing the embedded systems so all the architectural specification is analyzed in the paper.

General Term
Software Systems, Software Engineering, Software Testing, Embedded Systems

Keywords
Embedded Software System, Software Testing, Rational Test Real-Time (RT-RT)

1. INTRODUCTION

Embedded systems are in every “intelligent” device that is infiltrating; they do not provide standard computing services and normally exist as part of a bigger system. Embedded systems are usually constructed with least powerful computers that can meet the functional and performance requirements [1]. Embedded systems generally use microprocessors that contain many functions of a computer on a single device. Linux and windows Embedded are two popular operating systems for implementing embedded systems [2].

Most, if not all, embedded systems are “real – time”. A real – time system is one in which the correctness of a computation not only depends on its logical correctness, but also on the time at which the result is produced [3, 4].

In the recent years, the functions added to the embedded systems have grown, which increases the complexity of a system even more with more the development time and costs [5, 6]. Even though the embedded systems are real life applications, and real – time systems often works in an embedded scenario and are important to our daily life, so the production is increasing very enormously [7].
In this paper, certain issues are proposed that affect the embedded world with a large wallop [10]: These issues greatly affect the testability and quantifiability of an embedded system. Recent studies show that more than 60 percent of projects involved in embedded systems are late, even if giving more than 50 percent time in testing [8, 9].
In section 2, related work introduced in brief. And then, techniques involved in embedded software testing are presented. Moving one step ahead, we will also examine what makes embedded systems so difficult to develop and to test. Finally in section 3, we have given RT – RT [13].

2. DEVELOPMENT LIFE CYCLE OF EMBEDDED SYSTEMS

Embedded systems are real time applications and are implemented with an assortment of software and hardware. Hardware is to carrying out the action and software is to run the application effectively, depending on the different types of constraints like time, size, power consumption, reliability, and costs. [20] Ceremonious methods for contriving embedded systems start from stipulation. The stipulations are written for hardware and software. The problems with ceremonious methods are the lack of unified hardware – software representation and the immatureness of well – defined design of hardware and software. The SDLC for embedded systems is similar to the standard SDLC except the architecture, in embedded systems, the most suitable architecture is object oriented [19].

Most of the embedded systems designed are life or safety critical system. Life or safety critical systems are the systems where human safety is dependent upon the correct operation of the system. A system is a safety critical system if a failure can result in loss of life, injury or illness, serious environmental damage, significant loss of, or damage to, property, failure of an important mission. The basic system safety goal is to eliminate all single-point failures that could lead to unacceptable consequences and minimize the probability of accidents caused by multi-point failures [12, 13].
However, safety must always be considered with respect to the whole system, including software, computer hardware, other electronic and electrical hardware, mechanical hardware, and operators or users, not just the software element [14].
Safety critical software has been traditionally associated with embedded control systems.
As awareness of how systems can impact safety has developed, the scope of safety critical software has expanded into many other types of systems [28, 29].

An obvious example of a safety critical system is an aircraft fly by wire control system, where the pilot inputs commands to the control computer using a joystick, and the computer manipulates the actual aircraft controls. The lives of hundreds of passengers are totally dependent upon the continued correct operation of such a system [30].

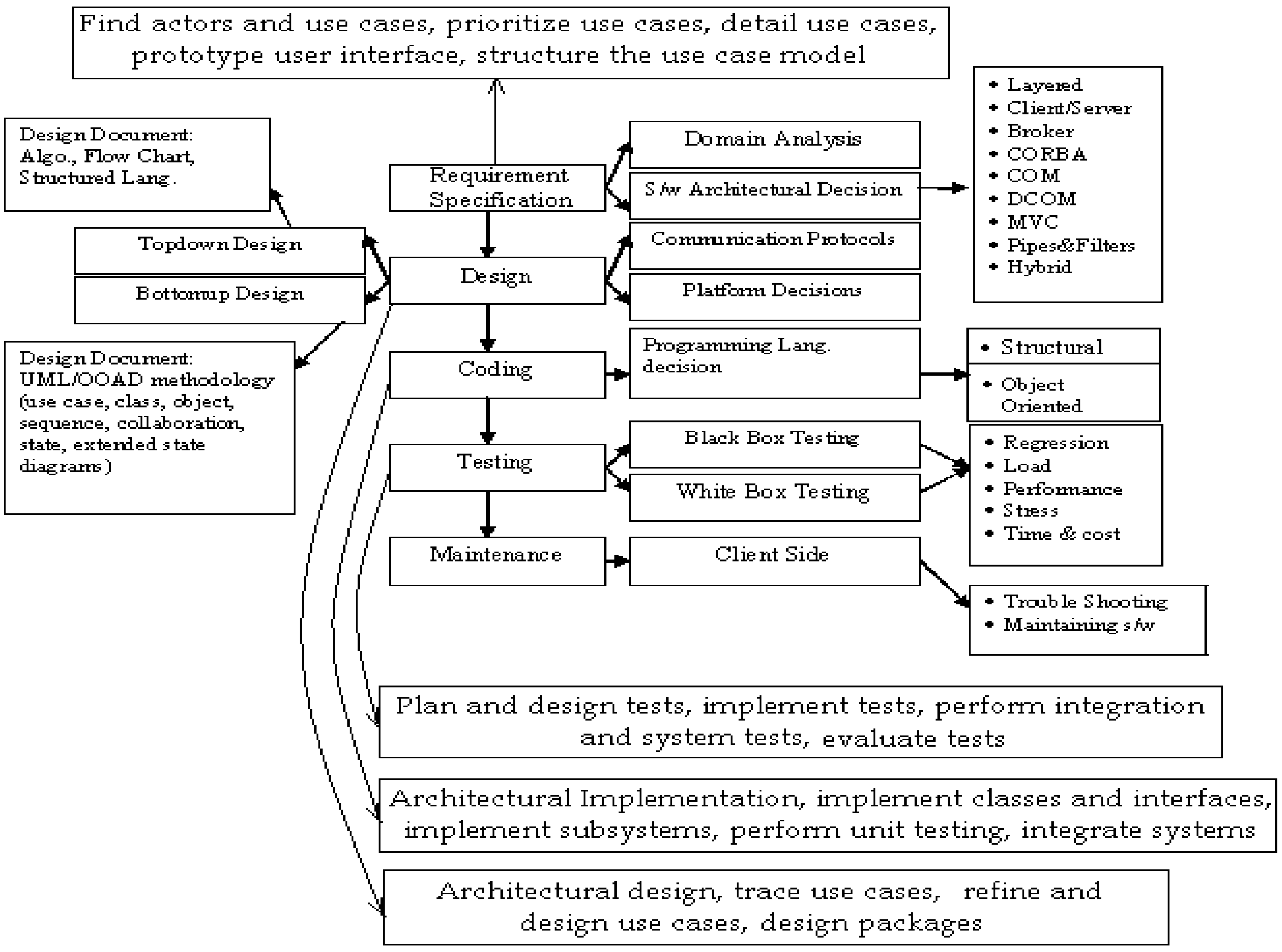
Find actors and use cases, prioritize use cases, detail use cases, prototype user interface, structure the use case model
Design Document: Algo., Flow Chart, Structured Lang.
Requirement Specification
Domain Analysis
S/w Architectural Decision
Layered
Client/Server
Broker
CORBA
COM
DCOM
MVC
Pipes&Filters
Hybrid
Topdown Design
Bottomup Design
Design
Communication Protocols
Platform Decisions
Design Document: UML/OOAD methodology (use case, class, object, sequence, collaboration, state, extended state diagrams)
Coding
Programming Lang. decision
Structural
Object Oriented
Testing
Black Box Testing
White Box Testing
Regression
Load
Performance
Stress
Time & cost
Maintenance
Client Side
Trouble Shooting
Maintaining s/w
Plan and design tests, implement tests, perform integration and system tests, evaluate tests
Architectural Implementation, implement classes and interfaces, implement subsystems, perform unit testing, integrate systems
Architectural design, trace use cases, refine and design use cases, design packages

Figure 1 - Development Life Cycle of Embedded Systems

2.1 Scope

This paper focuses mainly on the following issues which gave rise to many controversial talks in the embedded world:

a. Difference between the development environment and the execution of development.

b. A wide range of deployment architectures.

c. Lack of clarity in design models.

d. Tight resources on the execution platform.

e. Variety of execution platforms, which increases the cross – development environments.

As the Embedded software developer reaches higher and higher levels of productiveness, software testing must be applied to each and every step to deliver correct and validated software and if these issues eradicated during the whole testing process, the system will be a "lineament".

3. GENERIC TESTING TECHNOLOGY (GTT)

After the implementation, the most important activity which is carried out is testing phase. Effective software testing before release is crucial for product success. Based on the new metrics and an associated methodology for in – process validation of test case effectiveness, GTT is much more important.

Embedded systems need exhaustive testing, and these complex systems, test cases are critical for effective testing. However, the mere fact that testers use test-case specifications does not guarantee that systems are sufficiently tested. Numerous other factors also determine whether testers have performed well and whether testing was effective.

Software testing for the embedded systems is little difficult task then the traditional software testing because in embedded systems programming is very near to the hardware. Most of the systems are written either in assembly or machine language which are very difficult to test and debug.

4. DECIDING HOW LONG TO TEST AN EMBEDDED SOFTWARE

Considering the safety criticality of the system under test, the testing can be stopped based on subjective criteria and a reactive assessment of "Quality", for embedded systems the quality remains very high. But here we can take probability and utility model for software testing [3, 4, and 5]. This model

describes the number of bugs that are discovered by an arbitrary time or equivalently, the number of times between failures of the software.
Let the software fails stochastically under the process of testing.

Let $\{N(T), T \geq 0\}$

Where, N = Number of time the software fails
T= Time taken for execution

Thus,
$$E\{N(T+\Delta T)\} - E\{N(T)\} = b\,[a - E\{N(T)\}]\,\Delta T + O(\Delta T) \quad \ldots\ldots\ldots\ (1)$$

Where a = bug
And the equation,

$$\prod (a, b|T, N, n, \text{s♀}, \ldots\ldots sn)$$

$$= \frac{\prod (a, b)\, P\{N(T) = n, \text{s♀}, \ldots. sn|a, b\}}{\int\prod (a, b)\, P\{N(T) = n, n, \text{s♀}, \ldots\ldots sn|a, b\}\, da\, db}$$

$$= \frac{a^{n+\tau-1}\, b^{n+\alpha-1} \exp\{-(1+\lambda+\mu+\sum_{i=1}^{n} Si)\, b + a \exp(-Tb)\}}{\acute{\Gamma}(\tau+n) \int_{0}^{\infty} g(b)\, db}$$

where,

$$\frac{b^{n+\alpha-1} \exp\{-(\mu + \sum_{i=1}^{n} Si)\, b\}}{g(b) = \{\lambda + 1 - \exp(-Tb)\}^{\tau+n}}$$

2.1. *A Utility Function for Testing and Release*

The main functionality of this function is to describe the costs and benefits to the tester of the testing process

$$F(T) = (S + M + R)\, T$$

$$= FT \qquad \ldots\ldots\ldots\ (2)$$

Where 'S' is the staff cost, 'M' the machine cost and 'R' the lost revenue per unit of time.

Logically 'S' means Architects, Designer, Developers, Testing team and implementers where as 'M' denotes cost of licensed software, hardware cost, network protocols, and infrastructure cost and 'R'denotes old versions, beta release and new up gradations.

So, the utility function [6] for testing software in time T, in which N (T) bugs are discovered and corrected, followed by release: [6]

$$\mu\{T, N(T), Ɗ(T)\} = A - C\,N(T) - D\,Ɗ(T) - F(T). \quad \ldots\ldots\ldots\ldots\ldots\ldots\ (3)$$

5. REQUIREMENTS FOR A GTT

a. Provide a test case notation.

b. Provide alternative ways to implement to implement test cases.

c. Support test case deployment and execution.

d. Report observation.

e. Auditing security, success, and analyze failure.

6. ARCHITECTURE AND STRUCTURE OF COMPLEX EMBEDDED SYSTEMS

Embedded Systems are made up of extremely diverse architectures. Most of them work on a Real Time Operating System (RTOS) [7, 8]. Embedded software is a little bit different from the application software i.e. running on a computer such as in local machine. The target processor of an embedded system is typically minimal in function and size because its main goal is to reduce the manufacturing and production cost. Therefore the program is developed first on the local machine and then cross compilers are used to generate the targeted cipher. Some of the examples in embedded systems are consumer electronics, telecommunications, automobiles and controlling of industrial plants. Different domains have common structure in functional configuration.

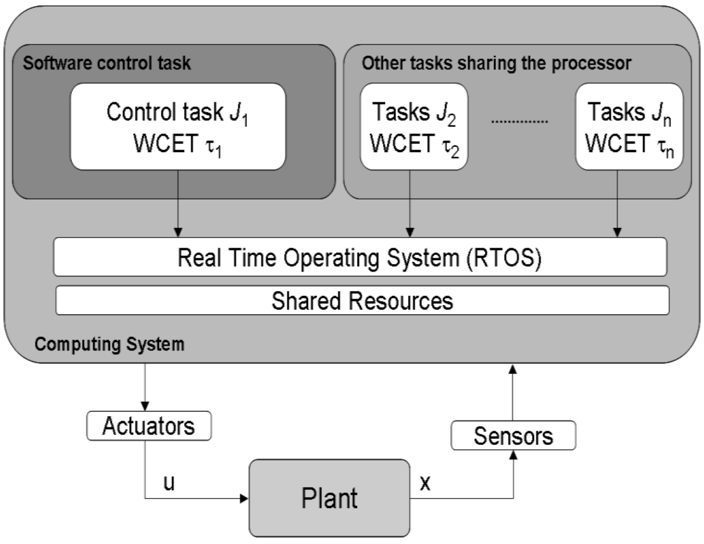
Software control task
Control task J_1 WCET τ_1
Other tasks sharing the processor
Tasks J_2 WCET τ_2
Tasks J_n WCET τ_n
Real Time Operating System (RTOS)
Shared Resources
Computing System
Actuators
Sensors
u
Plant
x

Fig. 2. A typical embedded software system.

RTOS and device drivers are closely coupled with hardware platform. In an instance for a particular application, a processor must meet a minimum speed, and the memory system must meet a minimum size.

To ensure high confidence in these systems, rigorous analysis is required before deployment. However, it is often infeasible to perform analysis on the actual system due to its scale and complexity.

Panoramas of Issues in Complex Embedded Systems which Affect Testing Process and the proceedings to exterminate them:

7. SEPARATION BETWEEN THE APPLICATION DEVELOPMENT AND EXECUTION PLATFORMS

An embedded system is any software system that must be designed on a platform different from the platform on which the application is intended to be deployed and targeted. By platform, on the development side, one typically means an operating system such as Windows, Solaris, HP-UX, or Linux. It should be noted that the percentage of UNIX and Linux users is much higher in the embedded systems domain as compared to other IT systems domains.

To cope up with this dual platform issue, the testing tool must provide access to the execution platform from the development platform in the most transparent but efficient way possible. In fact, the complexity of such access must be hidden to the user.

7.1 A wide range of Deployment Architectures

The first phase of designing embedded software is the software architecture design. Software architecture is the overall system structure as described by the components and connections among components [9]. Software architectural styles categorize architectures based on characteristics specific to a structural composition, such as shared data, abstract data type, implicit invocation, and pipe and filter.

The software team understands the target system at this phase and reviews it with the proposed hardware architecture. To compose target system architecture, the user selects the optional features desired for the target system. Once all the features have been selected for a target system, the target system architecture is composed from the corresponding architecture patterns [9].

7.2. Lack of clarity in Design Models

Model based approach has been advocated for design and analysis of these complex systems in order to produce confidence in the design and reduce development costs. In this approach, representative models of the system are judiciously used to predict its behavior and analyze various properties. Hybrid automaton [10, 11, 12] has been used to model and analyze embedded systems in which discrete and continuous components are tightly coupled.

In order to automate the analysis of hybrid automata, algorithmic approach has been developed. Algorithmic approach can be classified into two categories: reductionism methods and symbolic methods [13].

The former reduces the infinite hybrid (discrete and continuous) state space to an equivalent finite bisimulation and then explores the resulting finite quotient space, while the latter per- forms direct exploration of this infinite state space.

Even though the reductionism method based algorithms are guaranteed to terminate, the classes of systems to which they can be applied are very limited. Therefore, symbolic method based algorithms are generally used. Various computation tools with vastly different implementations have been developed for symbolic method based analysis. For example, d/dt [14] computes reachable sets by approximating reachable states based on numerical integration and polyhedral approximation; whereas the Level Set toolbox [15], which applies the level set methods [16], computes the evolution of a continuous set by solving the associated partial differential equation on grid structure.

Due to these implementation differences in computation method, data structure as well as analysis purpose, designing new analysis algorithms by using or modifying existing tools becomes infeasible or inefficient. Furthermore, designing a common interchange format [17] for these tools is difficult.

In order to resolve the analysis problem, the computation platform called Reach Lab is designed to enable

a. Separating the concern of algorithm design for analysis of hybrid automaton model from any specific computation implementation.

b. Separating the design of algorithm from specific hybrid automata- ton model so that the same algorithm can be reused for other system models. Reach Lab is developed based on the Model Integrated Computing (MIC) [18, 19] approach.

MIC approach is based on models and automatic generation of useful artifacts. In this approach, models are used not only to design and represent the system, but also to synthesize and implement the system using a modeling language tailored to the needs of a particular domain. These modeling languages, termed as Domain Specific Modeling Languages (DSML), have necessary constructs to allow the capture of useful information of a system as model particular to that domain. One can perform system analysis on this model. When this modeling capability is augmented with the capability of model transformation, even automated synthesis of other design models, and generation of executable system can be performed [19].

Keeping in view the safety of Hybrid Automation Model, we need to design its hybrid automata model in the system aspect and design the algorithm in the programming aspect.

The entire process can be summarized into three basic designing steps:

a. *Obtaining system model and algorithm specification*.

b. *Design phase of the system model:*

i. A hybrid automaton is drawn in the system aspect with discrete transitions connecting discrete modes.

ii. The designing of analyzing the algorithm which is hierarchical in nature, is modeled in the programming aspect.

iii. Stipulations of the computation input parameters to the algorithm and computational parameters have to be specified before translation.

c. *Implementation phase:*

Translators are used to convert the designed models into implementation for a certain computational kernel

7.3. Tight resources on the execution platform

Embedded system has limited resources. The technology used by Rational Test Real-Time involves embedding the test harness onto the target system.

This is done by compiling test data previously translated into the application programming language (C, C++ or Ada) within the test harness, using the available cross-compiler, and then linking this test harness object file to the rest of the application. This building chain is made transparent to the user by using the Rational Test Real-Time command line interface in make files.

8. VARIETY OF EXECUTION PLATFORMS, WHICH INCREASES THE CROSS – DEVELOPMENT ENVIRONMENTS

The Cross-platform development means that the development is done on a different platform (called the source or host platform) than the one on which the system will actually be run (called target platform). For example, a system is developed on Windows NT and it is then downloaded onto a custom hardware running a separate. RTOS, for the purpose of testing.

Cross-platform development brings issues related to differences in the source and target environment. The developers should develop the system as per the facilities available on the target environment and not what is available on the host. For example, the target RTOS may not provide all standard C/C++ libraries, which are otherwise available on a typical Windows/Unix setup, so such libraries cannot be used. Further, the code has to be built (compiled and linked) for the target environment.

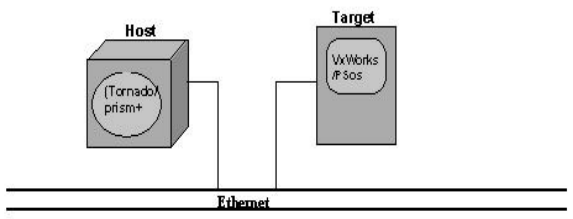

Fig.3. Cross Platform Development: [10]

In other words we can say that the execution application can range from a small micro-controller to a large distributed and networked system. It is increasingly common that multiple platforms are used within the same embedded system. From a development perspective, this kind of environment is referred to as a “cross-development environment”.

The large variety of execution platforms implies the availability of a correspondingly large set of development tools such as compilers, linkers, loaders, and debuggers. A Rational Test Real-Time Target deployment for a new target platform is usually achieved in less than a week, often within two days [10].

9. CONCLUSION

More the functions in the embedded systems more will be the complexity, which increases the development costs and duration. This gave us grounds to have a passable design and techniques to test embedded software. Embedded systems are difficult to test, because embedded systems are usually developed on custom hardware configurations, tools that is applicable to one may not be applicable to another application. So to exterminate the issues in the embedded world, I have canvassed these issues to develop high quality embedded software.

10. REFERENCES

[1] Testing embedded systems: Paul Szymkowiak, www.embeddedsystem.com

[2] Dinesh Kumar Saini and Nirmal Gupta “Fault Detection Effectiveness in GUI Components of Java Environment through Smoke Test”, Journal of Information Technology, ISSN 0973-2896 Vol.3, issue3, 7-17 September 2007.

[3] Kevin Laoghaire and Simon P. Wilson “Deciding how long to test software” The Statistician (2001) 50, Part 2, pp. 117 – 134.

[4] Singpurwalla and Wilson, Software reliability modeling: Statistical methods in Software Engineering.

[5] Goel and Okumoto: Singpurwalla, to determine an optimal time interval for testing and debugging software. IEEE Trans. Software Engineering, 17, 313 – 319.

[6] Goel, A.L. and Okumoto, K. time dependent error detection rate model for software reliability and other performance measures.

[7] Dinesh Kumar Saini “Testing Polymorphism in Object Oriented Systems for improving software Quality” ACM SIGSOFT Volume 34 Number 2 March 2009, ISSN: 0163-5948, USA

[8] Mary Shaw and David Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 2010.

[9] Dinesh K Saini, Mustafa Hasan “Architecture and Classification algorithms for large P2P Digital Library” The Second International Conference on Networked Digital Technologies OpenConf system, Crez Republic Europe 2010.

[10] Albert Sangiovanni-Vincentelli and Grant Martin, “Platform-based Design and Software design methodology for Embedded Systems”.IEEE Design & Test of Computers, April-June, 2000, p.2-15.

[11] Testing Embedded Systems Rational Unified process (RUP): Paul Szymkowiak.

[12] Dinesh Kumar Saini, Wail M Omar and Sanad Al Maskari “Healthcare Collaborative Framework for Chronic Disease Management in Oman” proceedings of International Conference on Multidisciplinary Approaches to Diabetes Research and Health (ICMADRH-2010), India, pp 46-52.

[13] T. Henzinger. The theory of hybrid automata. In Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, (1996), pp. 278–292.

[14] J. Lygeros. Lecture Notes on Hybrid Systems. Cambridge, 2003.

[15] Dinesh Kumar Saini “Sense the Future” Campus Volume 1- Issue 11, Page No14-17, February 2011.

[16] T.A. Henzinger, R. Majumdar. A classification of symbolic transition systems. In Proceedings of the 17th International Conference on Theoretical Aspects of Computer Science (2000), pp. 13–34.

[17] E. Asarin, T. Dang, O. Maler. The d/dt tool for verification of hybrid systems. In Computer Aided Verification, (2002), vol. 2404 of LNCS, Springer-Verlag, pp. 365–370.

[18] Dinesh Kumar Saini and Hemraj Saini "Achieving Quality Through Testing Polymorphism in Object Oriented Systems,"3rd International Conference on Quality, Reliability and INFOCOM Technology (Trends and Future Directions), 2-4 December, 2006, Indian

National Sciences and Academics, New Delhi (India). Conference proceeding.

[19] I. Mitchell, J. A. Templeton. A toolbox of Hamilton-Jacobi solvers for analysis of nondeterministic continuous and hybrid systems. In Hybrid Systems: Computation and Control, (2005), pp. 480–494.

[20] Dinesh Kumar Saini and Hemraj Saini "Issues and Problems in Generation of Automated Test Data for Object Oriented Systems," 3rd International Conference on Quality, Reliability and INFOCOM Technology (Trends and Future Directions), 2-4 December, 2006, Indian National Sciences and Academics, New Delhi (India) Conference proceeding.

[21] S. Osher, R. Fedkiw. Level Set Methods and Dynamic Implicit Surfaces. Springer, 2003.

[22] A. Pinto, A.L. Sangiovanni-Vincentelli, L.P Carloni, R. Passerone. Interchange formats for hybrid systems: Review and proposal. In Hybrid Systems: computation and Control, (2005), pp. 526 – 541.

[23] G. Karsai, A. Aggarwal, A. Ledeczi. A metamodel-driven MDA process and its tools. Workshop in Software Model Engineering, (2003).

[24] G. Karsai, J. Sztipanovits, A. Ledeczi, T. Bapty. Model-integrated development of embedded software. In Proceedings of the IEEE, (2003), pp. 145–164

[25] "Fault Detection System Based on Embedded Platform" ETTANDGRS08 Volume2 ,2008

[26] R. Alur, D. L. Dill. A theory of timed automata. Theoretical Computer Science, 126, (1994), pp. 183–235.

[27] Dinesh Kumar Saini and Nirmal Gupta "Class Level Test Case Generation in Object Oriented Software Testing, International Journal of Information Technology and Web Engineering, (IJITWE) Vol. 3, Issue 2, pp. 19-26 pages, march 2008. USA

[28] Lakshmi Sunil Prakash, Dinesh Kumar Saini and Kutti N.S. "Integrating EduLearn Learning Content Management System (LCMS) with Cooperating Learning Object Repositories (LORs) in a Peer to Peer (P2P) architectural Framework" ACM SIGSOFT Volume 34 Number 3 May 2009, ISSN: 0163-5948, USA.

[29] Wail M.Omar, Dinesh K. Saini and Mustafa Hassan "Credibility Of Digital Content in a Healthcare Collaborative Community" Software Tools and Algorithms for Biological Systems in book series "Advances in Experimental Medicine and Biology, AEMB" Springer, Page No, 2010

[30] Dinesh Kumar Saini, Sanad Al Maskari and Lingraj Hadimani "Mathematical Modeling of Software Reusability" 3rd IEEE International Conference on Machine Learning Singapore, February 26-28, 2011.

[31] Dinesh Kumar Saini, Wail M. Omar "Software Testing For Semantic Service Oriented Architecture for E-Health Software Services" SERP'10 - 9th international Conference on Software Engineering Research and Practice (USA) http://www.world-academy-of-science.org/, P.No. 240-246.

[32] Hemraj Saini and Dinesh Kumar Saini "AN Automated Test framework for Java Application" ICIT (IEEE Sponsored) Rourkella-200