The Relevance of Clean Room Software Engineering Methodology for the Development of Embedded Systems

Prof. Chandra Prakash V,
Dept. of Computer Science and Engineering,
K.L.University,
Vaddeswram, Guntur District, India, 522502
vchandrap@rediffmail.com

Dr. Sastry JKR,
Dept. of Computer Science and Engineering,
K.L.University,
Vaddeswaram, Guntur district, India, 522502
drsastry@kluniversity.in

Bala KrishnaKamesh Duvvuri
Dept. of Freshman Engineering
K L University,
Vaddeswaram, Guntur District, India 522502
kameshdkb@kluniversity.in

Abstract—Embedded systems must be designed to achieve high reliability, throughput, and desired response. The quality of the embedded systems must be very high and at the same time the cost of development must be as low as possible. Criticality is another issue that must be considered when it comes to building safety critical systems. Embedded systems must be tested in real time and the cost towards testing must be as low as possible while at the same time must ensure high reliability. Most of the embedded systems are reactive systems and therefore must respond to the external events and the events are processed leading to the controlling of the external events. Clean Room Software Engineering (CRSE) methodology aims at delivering high quality systems through usage of various verification and validation models at different stages of development. CRSE also uses extensive formal frameworks, thus leaving no chance either for direct human errors or hidden system errors. CRSE includes a parallel path for testing the developed systems by using the Statistical Use Case testing models which takes very less time for testing while at the same time ensuring high level of reliability. In this paper, the suitability of the CRSE for the development of embedded system is presented and a refined CRSE life cycle model is proposed using which highly reliable embedded systems can be built.

Keywords: Clean Room Software Engineering, Embedded Systems, Life Cycle models, Quality Systems, Refined Methodologies

CRSE LIFE CYCLE MODEL

CRSE uses a Life cycle model which recommends Formal models for analyzing, designing, development, and testing of the software based systems. The Process Flow model recommended by CRSE is depicted in Fig 1. The functional requirements and the usage requirements are derived from the Requirement specifications. The functional requirements are used to model the internal and the external behavior of the system whereas the usage requirements are used for testing the system. CRSE recommends the development of the system in increments and every time an increment is added, all the life cycle steps are carried. The processes related to usage modeling and Software Engineering modeling are conducted in parallel and Statistical Use Testing is carried using the test cases generated through usage modeling. Fig 1 shows the process flow adapted in CRSE.

The software engineering model is built around three Box structures (Black Box, State Box, and Clear Box). The black box structures are used to build external behavior through the stimulus and responses that are derived from the functional requirements manually and mapped to the black boxes intuitively. Even the Hierarchical decomposition of the Black Boxes is undertaken intuitively. The verification of the coverage of all the functional requirements is undertaken by enumerating the combinations of stimulus-response requirements and finding whether a stimulus-response is meeting the desired functional requirement.

The state box structure models the internal behavior and models the state transitions that take place due to stimulus or event occurring in the external world. Each of the state transitions is mapped to state variables and the state variables
undergo change when state transitions occur. The clear box deals with the predefined program structure and the code that needs to be executed when state transitions take place in the system. The state variables are set with appropriate values and processed by using the standard logics. Fig 2 shows the step by step processing that takes place during the software engineering phase of CRSE life cycle model. After every stage of the life cycle, formal verification is carried to check the consistency of one stage represented by a model with the other.

![Diagram of CRSE Life Cycle Model](image)

CRSE takes care of the quality issues right through the life cycle of the system by providing separate processes that deal with quality. The extra processes are related to verification and validation and therefore ensure high quality systems. This makes a case to use CRSE for the development of embedded systems.

II. ES LIFE CYCLE MODEL

Embedded systems (ES) are basically reactive systems and they follow the behavior of Stimulus-response mechanisms. The ES must process the external events taking place externally and respond in terms of an output or controlling of the external systems through actuators. The external behavior of embedded systems can also be modeled through Stimulus-response models that are advocated by CRSE. Embedded systems must be highly reliable, fault tolerant and must respond fast to the events taking place externally. The embedded systems must be high quality systems and their internal and external behavior must be modeled like the way the CRSE proposed using the black box structures and State Box structures.

The embedded systems must be tested comprehensively considering hardware, software and the integration of both. The testing of the embedded systems is complex and takes enormous amount of time for undertaking testing. The cost that will be incurred for testing of the embedded systems is also very high. The effort towards testing of the embedded systems can be greatly reduced through the use of Statistical Use Testing (SUT) method advocated by CRSE. Development of embedded systems involve in specifying the Hardware and software requirements. The requirement speciation also includes the integration requirements in addition to the non functional requirements such as response, timing, throughput, reliability and availability.

The development of embedded systems is done using hardware software co design model as shown in Fig 3. In this models the analysis, design, and code generation related to Hardware and software is done in a sequential flow and the code is individually optimized by performing the response computations and also scheduling activities. The optimized code is tested individually and then integrated and tested again before the same certified. After confirming that the Hardware and software are functioning properly, the software is moved on to the hardware and the entire system is tested again.

III. A SURVEY ON MODELS USED FOR DEVELOPMENT OF EMBEDDED SYSTEMS

Survey on Models for carrying requirements Analysis of Embedded Systems

Fine tuning and refining of the specifications at the early stage of development is achieved through representation of the specification through formal languages and conducting the simulation. However capturing the specification through a formal language is complex. D. Gajski et al., have recommended creating a conceptual model using the functional requirements. The model is described by using the descriptive language. The model is validated by either using a simulation or validation model. The result of capturing the specification is the functional specification which does not specify any implementation details [1].

The main issue is presenting the requirements as a formal specification using a model. Several of the models exist to present the requirements. [2] Decomposes functionality into activities that transforms data and the dataflow between the activities. The Finite State Machine Model (FSM) represents the system as a set of states and a set of arcs that indicate transition of the system from one state to other when certain events occur.

The model “Communication Sequential Processes” (CSP) decomposes the [3] a System into a set of concurrently executing processes, each of which executes a sequence of program execution including the variable assignment, loops, branches and procedure calls. Another model PSM (Program State Machine) combines FSM and CSP by permitting each of the states to be defined by a set of concurrent processes.

Many of the languages available today such as VHDL, Verilog that describe a CSP model through their process and sequential statement constructs. The VHDL and Verilog languages are also useful when the model is to be a FSM model even though the languages do not support the representation of state transitions. The language Spec Charts
meets the characteristics of any of the Embedded Systems but can only transform a model presented as PSM Model. FSM models are more suitable for representing real-time systems requirements as they implement event response model and the Language “state charts” can be used to represent the model as the language implement state transitions effectively.

The functional requirements from the point of view of software are presented formally using MSC (Message Sequence Charts) ITU-T [5] which are basically use case descriptions. SDL (Software Description Languages) ITU-T [4] are also used for formally describing the requirements of the Embedded Systems. The MSC and SDL were extended to PMSC (Performance Message Sequence Charts) [6] and [7]. The specifications in these formal languages are then optimized and partitioned to derive a specification that will be synthesized to HDL (Hardware Description Language) and software implementation. The informal requirement specifications are transformed into formal specifications. The formal specifications comprise use case descriptions given in MSC and a full functional specification in SDL. The timing constraints and resource requirements are added to MSC to form PMSC. The PMSC are then translated to SDL leading to SDL*. The requirements stated by the user can be presented as a set of use cases, and the software objects that help in realizing the requirements through interaction between the objects. The interaction between the objects are depicted through UML artifacts such as Sequence Diagrams, Class diagrams, Object Diagrams, State Transition diagrams etc. The UML artifacts are extended with timing constraints to take into consideration of the real-time requirements of the embedded systems. The requirements that relate to the Hardware in terms of the objects, their integration and interfacing and the timing of the signals however are not taken into consideration as it is not possible to model them through UML Artifacts.

[8] Have recommended that requirements projected for the development of embedded systems be recognized as a set of patterns. A pattern provides common solution for common problems. If the requirements are identified as a set of predefined patterns, then their corresponding design components both Hardware and Software which are readily available can be selected and reused, thus quickly building the solution. [9] Have recommended the commonly used designed patterns. [10] have identified High Level analysis patterns that can be used to represent the models of business processes such as abstractions like accounting. [11] Have added more structure to the analysis patterns by focusing on the domain of cooperative work and collaborating applications. [12] Discussed the relationship between non-functional requirements and design patterns. [13] Have defined requirement patterns based on use case/event models. [14] Have described the process of identifying the generic requirements from use case scenarios. Many of the authors have identified software architecture patterns, database access patterns, Fault-Tolerant Telecommunication system patterns, patterns for distributed systems, design patterns for Avionics etc.

[15] Have proposed a method for designing of the real time embedded systems by combining the internal strengths of UML and formal strengths SDL. They have advocated a design methodology for designing of the real time embedded systems. The traditional way of designing the hardware and software is no more valid due to ever increasing the complexity of the embedded systems and many of the authors have advocated the Software and Hardware Co design. The process of designing the hardware and software is almost similar. However the implementation of the hardware and
Software may differ. Most of the embedded systems have data flow paths and control flow paths. Control path configures and schedules the data paths. The data paths and control paths can be defined independently in respect of both hardware and software. 15] Have addressed the issue of control, communication and synchronization issues at the conceptual design stage itself. They have used OO concepts to specify the hardware and software using UML 2.0. They have also used SDL to

A survey on Models related to Analysis and Designing of Embedded Systems

D. Gajski et al., Has recommended that the functional specifications are transformed into a set of requirements that could be directly implementable [1]. A set of system components are selected that fully describe the physical and performance constraints. The implementable functional specifications are portioned across the selected system components. Several of the alternate design strategies involving the pre selected system components are explored keeping in view of the performance constraints. This step is called as exploration. The initial specifications are refined based on the initial designs strategy chosen. The refinement is achieved through moving each variable into the memory and adds interfacing protocol between the components and adds orbiters to linearize concurrent access to single resource. The system description is generated which details various components selected for development into hardware blue print. A co-simulation is performed to verify that the system level description is in consonance with the functional specification. The system level description thus refined has all the implementation details properly stated as a part of functional specifications.

Hardware synthesis is carried to transform higher level descriptions into a structure of RTL components such as Registers, ALU, etc. The structure usually contains control logic and data a flow for executing the arithmetic operations. Such a structure is called the FSMD (Finite State machine with data path). The control logic controls register transfers in the data path and generates signals for communication with the external world. A high level synthesis carried for the processor component from a register transfer library containing micro Architectural Models such as ALU. The control logic and RTL components are synthesized using finite state machines or by using logic synthesis Techniques [16], [17]. The result of software and hardware Co-Design is an RTL description which contains C code representing the software and RTL level netlist for custom components.

Frank has proposed to carry System synthesis and implementation synthesis using SDL*. The system synthesis translates SDL* into an internal system model. The system model contains two graphs: a problem Graph (PG) describing the functional behavior and architecture Graph (AG) describing the prototyping platform. While the PG is derived from SDL* specification, the AG is loaded from the Library. For each possible problem node to a processing element of the AG, search is made for Hardware/Software implementation. After System Synthesis, the resulting system is translated back to SDL*. The SDL* specification is then translated to VHDL for the hardware module and C for software module. A tool called CoGen translates behavior SDL* processes into implementation description, and adds the SDL Run Time Support System (RTSS) to this implementation. The RTSS supports all functions of the computational model of SDL such as Communication mechanisms and Timers. CoGen also generates configuration description for implementation and measurement.

[18] have advocated design of embedded systems in terms of Time state charts, Class diagrams with state charts and extended sequence diagrams and generated Extended timed automata and real time Petri Nets. Petri nets are used to determine the schedule of execution of Embedded Tasks. The extended timed automata are then verified with schedules using model checking verification procedure. The components identified through class diagrams are then mapped to Task schedules and extended finite automata. The mapping is used to generate the code and then proceed for implementation. In this case, no due consideration to the hardware is made and the whole of the design is based on the generation of the code form the some of the UML artifacts which include Extended state charts, Class diagrams and extended Sequence Diagrams from which Petri Nets are derived based on which the task scheduling is generated.

[19] Have used UML 2.0 Profile for designing of the embedded systems. UML 2.0 Profile called TUT-Profile defined as a set of stereo types for extending the UML mete classes as well as design practices to describe applications, platforms, and mapping them. A set of UML properties that relate to Embedded systems design can be identified [20]. Several of the languages that describe the design can also be used in conjunction with UML 2.0 using it extensibility mechanism. The Extensibility mechanisms supported in UML 2.0 are used as the basis to develop TUT-Profiles. [20] Have presented a UML profile suitable for embedded real time systems specification, design and verification. They included in the profile the Hardware and Software Co-design. The profile presents extensions that define functional encapsulations, and composition, communication, and mapping for performance.

21] Have presented a GUI language for defining the specification. The language includes domain specific classifiers and relationships to model the structure and behavior of the embedded systems. They have introduced new building blocks to represent platform resources and services and presented proper UML diagrams and notations to model platforms at different abstraction levels.

[22] Have defined a methodology for model mapping during different development stages. The modeling is undertaken based on active objects, modeling rules that help transforming the modeling objects. The UML profile for schedulability performance and time is standardized by OMG OMG [24]. The profile supports the interoperability of modeling and analysis tools. The UML-RT [25] has defined execution semantics to capture behavior for simulation and synthesis. The profile presented capsules to represent system
components with internal behavior is represented as state machines. All these profiles contain features for utilizing UML in embedded systems. However the profiles do not have any models defined for combining the application with platform. TUT Profile [19] has included the stereo types required to define the applications and the platforms and their mapping. The entire system design is provided three different types of descriptions that include Application descriptions, Platform descriptions, and Mapping. Both the platform and application descriptions can be developed independent of each other. TUT Profile mainly concerned with the structure of the application and platform. The application is identified as a set of active classes with internal behavior. The platform on the other hand is seen as a parameterized library in UML 2.0. The profile considers the behavioral modeling through UML constructs. The TUT Profile classifies the components that relate to the application and the platform as a set of stereo types and also defines various rules with which the components can be accessed.

[26] Have narrated sequential design flow method and Hardware-Software Co-design method and discussed various disadvantages of these methods special related to decomposition of problem solving into Hardware and software. He has recommended another design method which uses both sequential and Co-design method and introducing another step that undertakes the Evaluation of all the software and hardware components form the point of view of reliability, response, cost etc. and create a Library of the cost effective components. These components will be used in each of the partitions that relate to hardware and software design and development in the parallel paths.

[27] Have recommended a dual Petri Net Based method for designing the embedded systems. Classical Petri nets have been used for modeling several classes of systems T. Murata [28]. A number of extensions have been made to the initial Petri net finding for inclusion of data flow with the nets. These Include Predicate Transition nets [29], [30] [31] [32], Extended Timed Petri Net(ETPN) [33]. The extensions have considered two separate paths for inclusion of data and control. While the control path is represented as Timed Petri Net (TPN) and the data path represented as directed Graph where the nodes are used to capture data manipulations and storage. The ETPN have been used for designing of the embedded systems. ETPN has been extended to allow modeling of Hardware/Software interface and software structure.

The communications between the Hardware and the Software are implemented making the data dependencies explicitly defined. [34] Have recommended hierarchical Petri nets that model systems at different levels of hierarchy. This model includes both time and Tokens are redefined to implement the dynamic behavior to deal with data and control. The model suggested by [33] ETPN could be used for synthesizing the Hardware as the ETPN implements the concurrency. However, data and control flow is not linked in this model. [34], [27] have recommended usage one graph to model both the control and the data paths. Embedded systems specification will have both control function and data operation. Its functionality can be recognized as transformation of data through logical functions [35]. However it is necessary to undertake the internal modeling of the embedded systems especially considering Both Hardware and software. In is important that the linking of controlling and Data flow be considered in addition to consideration of Hardware- Software design. [27] has proposed a design method that considers Hardware-Software co design along with control flow and data flow.

[36] has proposed a component based framework for the design of the real time embedded systems. The framework proposed by them involves a systematic hierarchy of generic components which can be used as building blocks that can be used by any of the embedded systems. Most of the techniques available in literature are largely based on informal methods and by undertaking the manual coding of the software. A formal frame work that uses prefabricated components will help computer aided development of the embedded systems. Several authors have proposed several open frameworks [37], [38], [39], [40] [41], [42], D. Isovic and [44], [45] but the frameworks did not address great many design issues.

A software Framework can be defined as consistent set of executable models specifying relevant aspects of system structure and behavior within an application domain. The system model is inherently object oriented. It is conceived as a composition of interacting components such as function blocks, port based objects etc. which are mapped to real time tasks [40] [43] or alternately executed under static schedule [43], [45]. Most of the definitions that exist in the literature to define the components are informal and no standard as such exists to define the Components that relate to the embedded systems.

Many authors have recommended the hierarchal presentation of the components that are derived from the areas of control engineering and systems sciences rather than identifying the components based on the human experience and intuitiveness. Some of the standards like IEC 61131-3 have defined the standard components for signal processing and control functions however no recommendations were made regarding the reconfigurable state machines and function blocks. However several of the issues must be included in a framework which include the communication between the components, the interfacing of the components, Interaction patterns (Client-Server, Producer-Consumer, Publisher-Subscriber), port based objects having links with Physical hardware ports and various types of inputs and outputs, mapping of the buffers to the ports, Function blocks that configure and implement the control tasks through the specification of the control flow and data flow must be addressed. The control engineering approach must be extended to system engineering concepts. [36] Has proposed a framework called COMDES that address the above mentioned issues while forming the framework which can be used for designing the modern and complex embedded systems.

[22] Has proposed a method of undertaking the design and development using the prototyping of the components and using the same to validate the user requirements quite early in
the development of embedded systems. The based system engineering when done using the prototype components will help develop the embedded systems in the shorter period of time. [22] Has proposed a designed methodology addressing both issues of prototyping and the Component based system engineering with the help of UML models. [46] Have recommended a hardware software co-design that considers both A periodic and Periodic Specification of Embedded Systems. The design approach recommended by them considers both Soft real rime and hard real time requirements and considered concurrent occurrence of Periodic (Regularly occurring tasks in a sequence) and A periodic (UN predictable occurrence of a Task sequence) along with the Hardware. The co-synthesis suggested by them considered both the Software (Periodic and A periodic) and the Hardware.

[47] Has suggested embedded Object Architecture [EOC] for the development of embedded systems. They have recognized hardware devices in which the software is embedded and have recommended such pre packaged units. When an embedded system is to be built the prepackaged units that are needed are selected and integrated. EOC thus helps in quickly building the systems and that in incremental and iterative manner. EOC helps in easy prototyping, carrying modifications to the existing embedded system, and it enables the incremental development of devices, which is Impossible with traditional approaches used for developing the embedded systems.

Embedded Real Time Systems deal with complex controls Schemes and many concurrent processes that drive the controlling devices. The controlling mechanism requires precise timing of the controlling action. Static analysis help in detecting the timing errors if any, much more precisely than testing. If the systems are designed using models that it becomes easier to verify the system as precise semantic representations suitable for verification can be implemented.

Model checking is used [48], [49], [50] to undertake the verification of the Embedded Software. Model checking uses state based analysis that can show if the system satisfy a formal property for which formal specification exists. The UML models are mapped to ETA (Extended Timed Automata) [51] and OCL (Object Control Language) to TCTL (Timed Computation Logic) [52] Formal ETA models are generated automatically from user specified UML models by using a flattening scheme that transforms each state chart into a set of one or more ETA. Clock variables and constraints appearing in the state charts can be directly copied ETA. TheETS are then merged with the scheduler and state graph is generated. A model checker called SGM (A state graph Manipulator) [53] is used to verify system defined properties including dead states. Deadlocks, live stocks and user defined properties defined in OCL.

Failures in Mission critical and safety critical systems such as nuclear reactor systems can cause havoc and lead to costly rectification and the failures must be avoided [54]. Analysis and design techniques such as inspections, walkthroughs, and design reviews can be used for identifying the potential inconsistencies during the requirements and design phases of

the product development [55], [56]. However software testing is one of the critical phases of the product development where the product is checked for its compliance with its requirements through its execution [59].

D. Gajski et al., have proposed the use of simulation methods for the verification and validation of the functional requirements of the embedded systems[1]. Testing of the embedded system again is a complex process and [61] have emphasized that the embedded systems must be tested comprehensively considering Hardware, Software and both. Several of the testing methods must be used which include scaffolding, Assert Macros, simulators, emulators, third party tools, logic analyzers, monitors and the testing must be conducted considering the HOST where the software is developed, Target (Hardware Board) and Both. The comprehensive testing of the embedded systems is expensive and time consuming even though highly reliable and exhaustive.

The Co-design of hardware and software stems from the requirements engineering where the requirements are modeled and the models are converted to specification languages which are then used either for simulation or optimization. The basic issue of END-TO-END processing requirements of the embedded systems has never been addressed. No specific emphasis is made either for external or internal behavior modeling. The only verification mechanisms available are either through simulation or carrying comprehensive testing of the embedded systems. Even though formal mechanisms are used for Software and hardware co synthesis, no specific emphasis is made for the behavioral modeling. No formal frameworks are available for verification and validations of different stages of development proposed by the methods discussed at section 3.0. The methods also do not deal with verification of completeness of the requirements of an embedded system.

Development of High Quality Embedded systems requires that the verification and validation is carried at every stage of the development or co synthesis of hardware and software, the completeness of the requirements of the embedded systems and testing of the embedded systems be carried as fast as possible and with least cost. The embedded systems must be modeled from different perspectives and verification of one perspective with the other be carried. The development of the embedded systems must consider most important issues such as response time, criticality, concurrency and scheduling of the tasks using which the ES applications are built.

Survey on Code Generation related to embedded System

Code is to be generated out of the state model specification using which the internal behavior modeling is undertaken. Several authors have proposed methods and frameworks to generate code. [62] have presented a model based code generation framework for embedded real time system considering multiple operating systems, communication mechanisms, different hardware sizes and dynamic structures of the software. [63] A number of design and code generation
approaches have been presented in the literature for development of embedded systems that require implementation of concurrency. [64] have presented Framework called StreamIT which is based on dataflow formalism. Simulink [65], Real time workshop (RTW) from mathwork considered code generation based control system modeling and time has been considered as integral part of the model development. [67] have proposed partial evaluation methods for that help automating the designing and code generation process. [63] have used partial evaluation methods for optimized code generation by transforming a generalized actor based model to a target code while preserving the models semantics. [63] has generated C Code based on the Models. The code is is related to software that is embedded into software. The issues related to co existence of hardware and software has not been considered. Models are developed using the Graphic software and the Models are analyzed to generate execution context (Data Types, buffer sizes, schedules, parameters, model structures etc.). The partial evaluator considers the output generated by the model Analyzer and also a library of highly optimized target code and generates highly monolithic code that can read the input and produce the desired output. The model based methods so far has not considered any issue of generating code that is related to modeling the Hardware. The methods discussed so far just tries to generate code for control flows only.

[68] Have proposed a method of generating SystemC TLM Model from UML specification. Using the SystemC TLM Model, SystemC RTL Model and the embedded software is generated. The SystemmC RTL model is synthesized to generated FPGA RTL code in the VHDL language and a ASIC netlist is generated. The embedded software refined by adding RTOS level constructs into the embedded software. They have developed an equivalence of SystemC Components with UML notations and conventions. The authors while have explained the way SystemC code is generated using the UML constructs did not explain the way the embedded Software is generated.

[69] Have proposed method to generated embedded software from state diagrams. Prior to the generation of the code the state charts are validated for correctness related to unused states, one initial state, states with outgoing links, reachability of every state from initial state, existence of at least one final state, availability of at least one path that reaches the final state even if a loop is in existence at any of the state. A data set is recognized from the state machine which includes states, transitions and state conditions. These details are converted into XML state machine conventions. The XML state machine structures includes mapping of functions and task periods. XSLT (Extensible style sheet Transformations) are used for mapping functions and tasks to XML structures. The XSLT process converts the XML and XSLT files and generates code in any of the prescribed language. Here again the Code generation related HW and embedded Software has not been properly addressed. The code generation process includes converting the XML and XSLT documents into trees and mapping the tress. Both trees are transformed as stated into the XSLT document into any types of output file.

[70] Have investigated the fitness criteria for a programming language that can be used to generate the Embedded Software Code from model based specifications. They have checked the fitness of Java language from the point of object oriented semantics and the ability to support the issues related to concurrency. They have concluded that the limitation of the programming languages do not severely effect the code generation when model based specifications are used as inputs. [71] have used SDL (Specification Description Language) for modeling complex real-time embedded system. SDL is a Graphic and formal language. User can interact with SDL and build the embedded system from the scratch. The SDL language can be used to generate an executable application. No source code is generated into any of the Language. Issue like performance optimization, task scheduling etc. which are extremely required cannot be addressed in this way of generating the code. The issues of hardware and software co design are cannot be addressed using SDL. [72] have proposed building a system not necessarily the embedded systems using the Colored Petri net models (CPN). The model is debugged and verified for its exactness in the application. The CPN tool is then used to generate the final executable but not the source code that further can be optimized. Here again issues like performance optimization, task scheduling etc. which are extremely required cannot be addressed in this way of generating the executable code.

[73] Have explained that an embedded system can be developed in terms of UML Artifacts. They have proposed that SystemC code can be generated by way of establishing the SystemC equivalents to UML notations. They have added some standard extensions to UML so that the UML notations can be mapped to SystemC constructs. The tool Rhapsody has been used for converting the UML to an intermediate design. The intermediate design is further converted to Systems constructs manually. The extensions to the UML which are required for constructing the SystemC constructs have been added in the language of Rhapsody. Rhapsody uses XML as the language in which the internal design of the system is represented. The XML document is then parsed to generate an abstract tree. A SystemC code generator uses the parsed tree and the readily made templates to generate the SystemC code. Here again only the Code required related to Hardware design and interfacing is developed. The embedded Software that delivers the control function has not generated.

[74] Have recommended a method to generate code given a set of Hierarchical state Charts. They have also explained the way the Hierarchical or concurrent processing state charts can be developed. They have proposed that verification and validation be carried on the state charts and the associated state space. They explained that the state charts can be implemented either in direct or indirect way. In the direct implementation the state charts are directly implemented whereas the in the indirect implementation the state charts converted into intermediate state space before the same is used.
for undertaking the implementation. The second strategy is particularly simple to implement especially when the application to be implemented is non trivial and does not have any complex functions. The state charts can be directly implemented using the tools such as Statemate and Rhapsody. The method proposed by the authors addresses the issues related to both the hardware and software implementation. The strategy of implementation proposed by them includes converting the state charts to concurrently executing state charts and each of the elementary state charts is either implemented through Hardware or software implementation. This means that state chart portioning be done in such a way that the elementary state charts are either implemented by hardware. However this brings in some more complexities especially when interlinking either the software components or hardware components. Here again the issues related to performance computations and scheduling the tasks as required by an RTOS have not been discussed.

Survey on performance optimization of Embedded Systems

Several authors have worked in the areas of performance optimization either related to Hardware or Software but not considering both HW and SW together and also have not considered the specific issues of optimization in relation to each of the sequence flow within the embedded system. Even the important issues of schedulability, delay times, latency times have not been considered for computing the response time.

[75] have explained that there can be several strategies that can be applied for optimization of the performance of the embedded systems. They have presented a framework that enables taking decisions on application of several performance optimization strategies. They have proposed optimization models, code models and resource models which are integrated for predicting the impact of applying optimizations. [76] have proposed a new Network processor related code optimization that uses Hardware multi threading support. The method reduces the overhead for saving and reloading register contents for function calls via the run time stack. [77] have presented the performance optimization for software implementation of floating point operations on systems like FPA (Floating point Acceleration), or Floating point Units (FPU). [79] have presented data flow algorithms for optimizing the code through partial dead code elimination, and parallel redundancy elimination. The code statements are either hosted or sunk based on the availability of functional resources, dead code elimination, and redundancy elimination, the path profile guidance prediction and speculation. [78] have proposed code reordering techniques for generating schedules in which instructions can be issued without a delay. Code motion is performed that hosts instructions above branches and some instructions are sunk below the branches. Some extra code is added to preserve some semantics. Some of the code optimization recommended by them includes partial redundancy elimination, elimination of dead stores. They have shown how the existing dataflow frameworks can be best utilized to effect the optimizations that are presented.

[80] have presented code optimization when code is generated using the models. The approach suggested by them includes building models, generating code from the models and compiling the generated code. They have claimed that modeling semantics that help optimizing the code are lost while the code is generated, thus making it difficult to generate the code. They have proposed a new level of abstraction that helps optimizing the code. Asam have presented that the optimization if embedded systems is constrained by the environment are resources in terms of memory, processing power, energy etc. They stressed that it is better to concentrate on the performance optimization at the code level. They have proposed addition of more number of abstractions to the models which help undertaking the optimization apart from undertaking optimizations at compiler level [82] have proposed a modeling language CHARON that models hierarchal specifications of interacting hybrid systems. They have explained how the code can be generated out of such models. They have however have not discussed any performance optimization issues. [83] have exploited the Retiming, algebraic and redundancy manipulation transformations, for performance optimization of the embedded systems in terms of latency and throughput. They have introduced simple modification to Leiserson-Saxe algorithm and introduced a new algorithm for handling tasks. They have introduced a new negative retiming techniques with both algebraic and redundancy elimination techniques for latency optimization. [84] Have presented a system for performance evaluation of embedded system based on code analyzer, testing agents, data analyzer and report reviewer. The performance of ES software is evaluated by introducing various types of code elements into the source code and running the same. The statistics thus obtained are used to optimize the same. [85]. Have presented analysis of source code considering a specific processor. They have used Bit-True data flow analysis is applied to the source code and code optimization is achieved in terms of saturated arithmetic, reduced bandwidth and exploit SIMD data processing within source code.

[86] Have explained that the re-targetable C compilers are widely used to provide support to the new processors exploration. The re targetable compilers have no machine specific code optimization. This problem can be circumvented by inventing flexible, reconfigurable code optimization techniques. They have re-targeted machines with SMID instruction support. They have proposed SMID code optimization techniques that are integrated with re targetable C Compiler. [87] Have explained that the exploitation of address generation units AGUS which are provided in DSPS is undertaken during the process of code generation in respect of DSP processors. They have presented a method of code optimization by exploitation of AGUS and integrate the same with code scheduling. They have used probability based methods for achieving global address assignments. [88] have proposed a novel feedback driven program optimization technique that profiles and determines the run time behavior of code segments in a program to find different patterns of behavior, correlated different run time behaviors of a program.
Software plays very important role of controlling the system in many safety critical systems through actuators. In-depth Testing of the software is necessary when higher levels of reliability and availability of the embedded system is critical. Software design, implementation and testing are performed iteratively with errors identified during test case execution. The errors are addressed by making changes to the software design and implementation and are retested. Validation of embedded system is equally important as it involves the hardware testing through fault injection techniques and testing of response time and throughput as a whole.

Embedded Applications belong to a different class of applications which throws several challenges especially related to Testing. The Software testing process to test the embedded applications involves testing individually Hardware, Software and both Hardware & Software together. The process of testing an embedded application is rather complex and needs a detailed study. The process of testing a loaded application however is more streamlined and several of the Tools such as Load Runner, WIN Runner, Team Test etc, are available using which different testing can be carried in a unified manner.

Development and testing of embedded software is especially difficult because it typically consists of large number of concurrently executing and interacting tasks. Each task in embedded software is executed at different intervals under different conditions and with different timing requirements. Furthermore time available to develop and test embedded software is usually quite limited due to relatively short lifetime of the products. Cost-Effective testing of embedded software is of critical concern in maintaining a competitive edge. Testing an embedded system manually is extremely intolerable of buggy embedded systems. Embedded system deals with external environment by way of sensing the physical parameters and also must provide outputs that control the external environment.

In embedded systems the issue of testing must consider both hardware and Software. The mall-functioning of hardware is detected through software failures. The target Embedded system does not support the required hardware and software platform needed for development and testing the Software, Hardware and both. The software development cannot be done on the Target Machine. The Software is developed on host machine and then installed in the Target machine which is then executed.

The Goals for testing the embedded Systems differs from that of the loaded systems. Testing of the embedded systems directly using target board will not realize all the Testing Goals of the embedded systems.

- Finding the bugs early in the development process is not possible as the target machine often is not available early in the development stage or the hardware being developed parallel to software development is either unstable or buggy.
- Exercising all of the code including dealing with exceptional conditions in a target machine is difficult as most of the code in an embedded systems is related to uncommon or unlikely situations or events occurring in timing relationship with other
- Difficulty to develop reusable and repeatable tests as it is difficult to create repeatable event occurrence sequence in the target machine
- Maintenance of audit trail related to test results, event sequences, code traces, and core dumps etc which are required for debugging cannot be maintained in the target machine due to primarily not having any disk storage in target machine

To realize the testing Goals it is necessary that testing be carried in the host machine first and then be carried along with the Target [89]. Embedded Software must be of the highest Quality and must adapt to excellent strategies for carrying testing. In order to decide on the testing strategy or the type of testing carried or the phases in which the testing is carried, it is necessary to carry on with the analysis of different types of test cases that must be used for carrying the testing

Every embedded Application comprises of two different types of tasks. While one type of tasks deals with emergent processing requirements, the other type of tasks undertakes the processing of input/output and also various processing related to housekeeping. The tasks that deal with emergent requirements are initiated for execution on interrupt. Test cases must be identified sufficiently enough that all types of
Embedded systems are event driven initiated due to a change in the physical environment as a consequence of a stimulus. Events occur in some sequence and there may be relationships existing between the occurrences of the events. Events also provide the end user perspective. Embedded system must be tested for proper processing of the events right from the stage of event occurrence due to a stimulus till the time the control action is taken. The testing of the event based processing can be considered as integration testing.

Events are processed by using patterns. Patterns are related to the timing and occurrence of prior and post conditions. A Pattern of processing may require the fulfillment of some initial conditions and some output conditions after the event has taken place. The prior conditions and the post conditions must occur as per the stated timings. The processing of the Events as per pre identified patterns is equally important. The testing of the embedded systems must be undertaken from END-TO-END testing which is a kind of integration testing. END-TO-END testing must be undertaken from the user perspective. It is the minimal integration testing required to ensure that events are processed as per the patterns attached to them.

Changes also take place after the embedded system is implemented. Changes may take place in Hardware or software or both. The Regression Testing of the embedded systems must also be undertaken every time a change is undertaken. Testing of the embedded systems must be undertaken either offline or online. Offline testing of embedded system is conducted by simulating the production system. Online testing is undertaken when the embedded system is connected to the production system. The testing should be undertaken either offline or online depending on the requirement.

The testing of the embedded system software can be conducted at the HOST using techniques such as Scaffolding, Assert Macros, or instruction set simulators. The testing of the embedded systems hardware can be carried using the test gadgets such as logic Analyzers connected to the Hardware and probing for the occurrence of various kinds of faults. The logic analyzers can be driven by HOST based software to which the gadget is connected. The test gadget in this connects the TARGET and HOST. The testing of the behavior of the production system when connected to the Embedded System can be carried along with the HOST machine. The primary strategy of testing should be that the testing be carried as fast as possible and as reliable as possible as the testing can be carried at different locations and using different methods.

Testing of embedded systems is complex. Testing of embedded systems must be done quite early in the development of the system. Development of the embedded systems involves analysis, design and development of both Hardware and software in individual paths by using V model. The testing of Hardware and software are done independently and then are integrated and tested together. Testing of the software has to be done considering the whole code instead of individual units to guarantee proper running of the software. The testing strategy must include various locations of testing, and also different methods used for the testing.

Comprehensive testing of the embedded systems involves conducting the Unit Testing, Integration testing, END-TO-END testing, and Regression testing. The testing process should consider all aspects of testing Hardware, software and both. Chapter 2.0 has detailed the test case types that must be considered and the applicability of the same for a particular embedded system must be verified and considered for undertaking the testing of embedded systems. The comprehensive testing of the embedded system involves the usage of several methods, tools, techniques and locations. [89] have proposed a test process architecture model for undertaking the integrated testing that will help in undertaking the comprehensive testing of the embedded system. Comprehensive testing of the embedded systems requires the usage of test cases and various Architectural Frameworks and models, tools and processes.

CRSE has not provided any direct methods using which testing of the embedded systems are undertaken. However some authors have used some of the models stated by CRSE for undertaking the testing of embedded systems. Thomas Bauer et al., [90] have used statistical testing method for undertaking the testing. He has proposed that the Usage model be constructed based on state transitions that takes place when a system is used. The state transitions are traced out of sequence based specification.

The usage graphs are updated with probability distributions. They claim that when each of the transition takes place state variables undergoes change and each of the transition can be represented by a script that considers the inputs and the expected outputs. All the scripts that are attached to the transitions in a path are concatenated to form into a test case. The test cases are fed to a tool which is a HW + Software solution to which probes are connected to the embedded system. The author has considered only hardware states and has not bothered about any control logic that is generally implemented using ES software. The author has no doubt used Statistical l methods but the model suggested by them is not suitable for testing the real time embedded systems.

[91] have presented a method for testing a Embedded product using usage models and statistical methods. They have shown the testing process for Radio communication based application. The Application chosen by them in non real time and as such no control logic exists. The usage states of the application have been manually recognized and have shown the way the usage and test chains can be generated by constructing Usage Model Transfer Matrix and Usage Model Excitation matrix. The measure of testing accuracy has been explained by way of correlating both the matrixes. The random test case generation using Transfer Matrix has been
explained. Here again only the usage states representing the hardware states have been considered and the test cases generated by this method does not support the testing of control logic. The ES testing models that are presented based on statistical methods are not quite suitable as they support the testing of the embedded systems from the Hardware point of View Only. [92] has recommended several models for undertaking the comprehensive testing of the embedded systems. He has recommended that Comprehensive testing of the embedded system is a necessity when testing of Hardware, software and both has to be undertaken using several distinct methods at different testing locations.

IV THE RELEVANCE OF EMBEDDED SYSTEMS TO CRSE METHODOLOGY

Clean Room Software Engineering (CRSE) development models rely on stepwise development and refinement. CRSE has built in Verification and Validation (V&V) processes that are undertaken after completing every step of development. Quality Assurance is directly built into development model. The development model of CRSE supports the incremental and iterative process of development. The main aim of the CRSE is to reduce the effort required to undertake testing. CRSE uses statically use testing assures the quality and certification of the Application.

Relevance of Hierarchical (Box Structure) Modeling

CRSE is based on Box structure way of modeling. Box structures are designed through hierarchical representation of the system using parent child relationships. Any complex system can be represented by way of tracing inputs to the system and outputs generated from the system, thus defining the boundary of the system within which the system should operate. The hierarchical structures help in systematic decomposition of any complex system. Every stage of development is achieved through three Box Structures. Analysis of the systems is achieved through (Black Box Structure), Designing of the system is achieved through (State Box Structure) and the code development is achieved through Clear Box Structure. Black Box structure models the functional decomposition till an elementary level of functional (requirement) identification is achieved. Each of the requirements is modeled through a stimulus (Input, External Event, and Internal Event) and a response (Output to a device or controlling a device). All the box structures defined by CRSE use the hierarchical structures for modeling either the internal or external behavior.

Embedded Systems must provide for END-TO-END (E2E) processing whenever an external event takes place till the time a control action is undertaken to control the external environment, meaning the entire interaction between the hardware devices, between the hardware devices and the Application Code and the interaction between the Software Components and in between the Application Code and Actuators through which the control function is achieved. The required processing can be presented hierarchically and thus can be clearly be represented by hierarchically models. Since development of Embedded systems are undertaken using the Hierarchical models, clearly creates a case for development of ES systems through CRSE Principles

Relevance with reference to external behavior modeling

CRSE believes in modeling Elementary level of Black Boxes using the formal methods that involve mathematical representation. Every stimulus is proposed to be processed using a function that produces the output. It has been proved in the literature that presenting software processing and the hardware interaction required to process a stimulus and produce output mathematically is complex procedure and sometimes not even feasible. However it has been proved in the literature that the processing of a Stimulus through a set of state transitions leading to a desired response is possible. The Latest findings of CRSE recommend that the elementary level of functional requirements be represented as a state diagram. The Black Box structures clearly models the external behavior of the system from the end user point of view.

Embedded Systems also can be categorized as stimulus response models. Every event that occurs in external environment must be processed in real time and a control action is undertaken if required. Embedded Systems undergoes several hardware and software transitions to process an event in the real time. E2E provides for minimal of processing required whenever an external event takes place. A single E2E can be recognized as a set of related functions which should be undertaken together. A single E2E processing is a complex thin thread derived from several simple thin threads. A thin thread provides for a simple processing to be undertaken when an external event takes place. Thus a Complex thin thread is similar to a Black Box Structure through which external behavior of a system is modeled. Every simple thin thread can be modeled through a stimulus response block. Thus Black box structures can be conveniently be employed for conducting the analysis of the embedded system through stimulus response models.

Relevance with reference to Stimulus sequences

Every system whether loaded or embedded produce a set of outputs after a sequence of stimuli are undertaken. The stimulus sequence may produce intermittent outputs which may be used as inputs when a stimulus takes place in a sequence. A set of stimuli that takes place in a sequence shall produce a final output. For any system there will be only few stimuli sequences that are valid which must be identified. A thin thread can be represented as a stimulus sequence which produces a specific output. The development of precedence relationships between the stimuli will clearly develop into all possible valid stimuli sequences. Thus the stimulus sequence models suggested by several authors in relation to modeling Black Box structures clearly can be used to model the external behavior of the embedded systems.

The thin threads that represent an embedded system clearly can be correlated as sequence stimuli, thus providing the basis of identifying the stimuli sequences through the knowledge of the thin threads which can be derived from the functional specification provided by the user. Unlike the loaded systems, one has to consider the transitions that take place due to
Hardware and also Software when it comes to the embedded systems. The stimuli sequence thus can include the Hardware stimuli and the Software Stimuli. The combined stimuli sequences cleanly deliver one or more of outputs. A directed graph (Mealy Machine) representing the entire system can be derived connecting all stimulus and responses.

Relevance with reference to Internal Behavior

The internal behavior of any of the system can be modeled through state transition of any system. Every time a stimulus occurs the system must be expected to transit from one state to other or at best transit to the same state again. Moving from one state to other can be related to the occurrence of a Stimulus. The mealy machine constructed out of the black box structure can be conveniently be used to construct the state machine automatically. The state machine clearly reveals the transitions that take place within the system due to occurrence of sequence stimuli in between the source and sink states.

Every thin thread can be identified as state transition path and a set of state transition paths together can be regarded as a Complex thin thread which completely provides for E2E processing. The processing that should take place as a consequence of state transition can be designed as entry and exit procedures. Each of the state can be related to set state variables. The exit and entry procedures call for the changes in the values of the state variable. Unlike the Loaded systems, embedded systems do consider the state transitions within the hardware and the software and both. Both type of transitions due to Hardware and software can be modeled into the state diagram. While the software transitions reflect on the values of the State variables, hardware transitions reflects on the state of signal transitions. The state transitions due to Hardware and software clearly fits into the state model. The ability to transit from the Black box structures to State Box structure automatically and the addition of procedures and data to each of the state or the state of the signal if the state belong to the Hardware transiting clearly support the step wise refinement advocated by CRSE. Moreover each of the transition can easily provide an insight into a testing step and each path in the state diagram can be considered as a test case.

Relevance of code generation through a Clear Box

CRSE clearly advocate automatic code generation through a process description language that can be related to the entry and exit procedures that are associated with stimulus occurrences. The PDL constructs with data passed can be converted into any language equivalent. Unlike loaded systems, in the case of embedded systems, the code generation is required only when the transitions are in between the Hardware and Software, or in between the Software components or in between the software and hardware components. In fact efficient code generation can be undertaken in the case of embedded systems as standard code can be generated for interacting with any of the device on the embedded board. Thus code can be generated automatically more efficiently in the case of embedded systems.

Relevance of Verification and Validation at every Stage of Development

CRSE principles lay stress on verification and validation at every stage of development. CRSE believes in prevention of defect during the Analysis, design and development than removal of errors through testing. At the Black box stage the completeness of user requirements cab validated using Stimuli sequences which are generated through stimulus precedence relationships. Care is taken to ensure that all the requirements are properly spelt out and the requirements are properly built into stimulus response models. The automatic development of the state models from Black Box models also ensures that the internal behavior is also built properly. Similarly the automatic generation of Code using the Entry exit conditions using the standard templates helps in generating the quality Software.

CRSE do not advocate conducting of testing in terms of unit, Integration, and Regression Testing as the CRSE principles stress on continuous verification and validation after every stage of development. Rather CRSE believes in Statistical Use Testing (SUT) for the purposes of certification. In STU a sample of test cases are considered for testing and the sample is drawn from the total population of the test cases. However in the case of embedded systems thorough testing of the hardware, software and the integration of both must be tested in addition to the SUT proposed by the CRSE.

The Relevance of Statistical Use Testing and the Reliability evaluation

CRSE advocates the Statistical Use Testing through the probability of usage attached to each path in the state Box Model. Probability of occurrences is attached to each path starting from sink. The paths emanating from a state node shall add to the total probability. Most frequently used paths are provided with more probability. The probability of usage of a path is estimated based on experience, or experimental results. The Random sampling Techniques are used to pick a sample of test cases. The sample of test cases picked shall have more test cases related to the paths that have been defined with higher probability values. SUT do not guarantee that every path shall be considered for testing at least once.

Embedded systems must be highly reliable. Failure of any apart of the system will completely break the entire system. The testing of the system includes testing of the hardware, software and both must be undertaken. Comprehensive testing of the embedded systems must be undertaken. Every path must be tested. In embedded system some of the paths are critical. Any errors occurs in these paths will jeopardize the entire system. Higher probability values are assigned based on the criticality of the path. Rigorous testing shall be carried to verify the critical paths. Thus SUT, in the case of embedded systems shall be carried not only for carrying the comprehensive testing but also to carry rigorous testing in the critical paths. The sample size chosen will be comprised of few cases in respect of non critical paths and more cases in respect of Critical paths. A genetic algorithm approach will be essential to test the neighborhood of the faulty sections of the system.

Thus the CRSE principle that SUT be used for Certification shall also be application but with more rigorous
usage take in to consideration of the criticality of the system. The reliability evaluation of the test results obtained after undertaking the testing will guarantee the certification of the system.

The Relevance of Incremental and Iterative Development

CRSE advocates that the every application be developed in increments and at every stage of an increment the complete life cycle of development is undertaken. Unlike this, the development of embedded systems requires that the whole of the system be considered right in the begging as the changes to the hardware are not generally recommended to the hardware. However the software can be developed in increments and the iterative model as suggested by CRSE can follow.

V A SURVEY ON MODELS USED FOR IMPLEMENTING CRSE AND ITS RELEVANCE TO MODELS USED DURING THE LIFE CYCLE MANAGEMENT OF EMBEDDED SYSTEMS AND RECOMMENDATIONS ON REFINED CRSE MODELS AT DIFFERENT STAGES OF DEVELOPMENT.

A number of models have been proposed in literature that is either related to design, or verification and validation related to Black Boxes, State Boxes and Clear Boxes. The discussion below addresses whether the models recommended stands for the basic objectives of CRSE and also their relevance to the Embedded Systems models which are discussed in section 3.0.

A. Refinement at Black Box Stage

[93] Proposed modeling Business problems in box structures where the users are required to identify the inputs (stimuli), outputs (responses), state, and processing needed by the problem. The box structure hierarchy supports referential transparency, state migration, transaction closure, and common services [94]. It forces the user to perform a complete analysis of the problem before it can be described exactly. It supports the division of complex problems into sub problems. Before going to the next hierarchy level, the clear box of the previous level must be specified to show the connections and data flow among the embedded black boxes which make the black boxes at the next level referentially transparent.

[93] Advocated development of a solution procedure by mapping the box structured problem model onto a set of potential management science solution techniques. They developed an abstraction level framework classifying the management science solution techniques to select an appropriate solution technique for solving business problems. The framework consists of four levels namely Objective Level, Structural Level, Instance Level and Solution Level.

[95] Have explained that the correctness of the sequence enumeration is verified by tracing every entry in the enumeration to the requirements which justify the entry. If no requirements address the specific entry, then one must document a derived requirement and trace the entry to the new derived requirement. The derived requirement must be approved by the domain experts and customers. The next step is to arrive at canonical sequences which are used in the derivation of the black box by adding a collection of condition variables. The complete enumeration is a complete, consistent and traceably correct black box. [96] Explored a new specification method for Clean room’s black box description in a precise, concise and readable manner. They used entity structure diagrams drawn from the Jackson Structure Diagrams (JSD) for writing their method of specifications. They formalized this concept using operators inspired from process algebras. [97] Extended the box structure theory of information systems development and utilizes Computer Aided System Engineering (CASE) tools, such as a Petri-net analyzer. [86] identified the decomposable parts of information systems that have coherent real time behavior in the system environment. The coherent parts accept stimuli in real time and produces responses accordingly. [86] have defined black a Box as a Mathematical function that maps a stimulus to a response and they advocated that stepwise refinement is necessary for identifying the ideal Black Box structures.

[97] have proposed to add an additional white box to address the issue of concurrent interacting systems. [101] have proposed extension of the domain of applicability of box structures by specifying software systems or system parts. The extension of semantics of Black boxes was needed to address the interaction between the black boxes. [102] advocated that the object-oriented analysis and design concepts can be applied to black box hierarchical structures. The black box specification encapsulates the behavior of the object and its interactions with other objects while hiding lower level details. Each black box object is designed as a part of an existing inheritance hierarchy or initiates a new inheritance hierarchy. [103] presented the concept of thin threads which are true representatives of the functional requirements of a system from the end user point of view. This model typically suits to loaded systems and can be used to present the processing requirements by using the hierarchical structures. The thin threads to be used as processing structures can be presented as a Tree structures. [58] have presented event patterns which together forms the basis for processing occurrence of a set of events in a sequence. About 80% of time is to be spent on integrated processing of the embedded systems as the integration of Hardware Devices.

Thin threads at each of the Level are grouped to depict different END-TO-END Views of the system. Thin threads are similar to use-cases [104] describing END-TO-END Scenarios. However, thin threads help in conducting dependability analysis, risk analysis, Coverage analysis, Traceability Analysis, Completeness and consistency checking and also form a type of Framework using which the test cases can be generated and testing is carried.

The models proposed in the literature especially the models that related to End to End processing can amply be used for modeling the embedded systems in terms of the black Box structures. [105] have proposed a formal framework for modeling the external behavior of embedded systems through black box structures. [106] have presented a Formal Framework for presenting the requirements specification of an Embedded System as a Black Box Structure.
Refinement for Verification of Black Boxes

[99] Have recommended the verification of the Black Box structures in terms of comparing the actual functions within the Black boxes with the intended functions. Here, the verification process is manual and, hence, becomes tedious and complicated when the Black boxes are represented based on mathematical foundations. [100] have extended the semantics of Black Box structures to accommodate the extra functionality required to build Concurrent functions within the Black box structures. They have shown the referential transparency of Concurrent Black Box (CBB) structures to Black box (BB) structures. Here again, the verifiability is limited from CBB to BB but they have not extended the verifiability to Functional requirements.

[108] Has proposed Failures-Divergences Refinement (FDR), a model checker for the systems developed using CSP (Communicating Sequential Processes) [5]. [95] have proposed that the correctness of the sequence enumeration is verified by tracing every entry in the enumeration to the requirements which justify the entry. If no requirements address the specific entry, then one must document a derived requirement and trace the entry to the newly derived requirement. The derived requirement must be approved by the domain experts and customers. The next step is to arrive at canonical sequences which are used in the derivation of the Black box by adding a collection of condition variables.

[109] [7] have proposed the development of Box structures using CSP language and used FDR for verifying and validating the properties such as completeness, connectedness, consistency, determinism and control laws that are specified in the system requirements. However, the verification of the requirements could be done to the extent that the requirements could be presented as CSP language equivalents. CSP, as such, has no built-in language constructs that are suitable for event handling requirements.

[110] have presented a framework for verifying and validating the functional requirements by adapting a two-step process which includes converting the functional specifications into a set of extended finite state machines and converting them into a specification amenable for model checking, consistency checks, and input & output checking. In this work different types of verification and validations are carried out with reference to the functional requirements. Inadequacies in functional requirements cannot be verified with this kind of approach.

CRSE suffers from lack of Formal Frameworks to verify Box structures with the requirements specification. [105], [106] have proposed Formal frameworks for verification and validation of Black Box structures from two perspectives END-TO-END processing, and Use case models. The verification mechanism is built around generation of stimulus-response sequences in two different ways and proving that the sequences generated are the same. Thus, the mechanism ensures that the system has been designed properly. CRSE has advocated the Formalism for modeling the external behavior and also for verifying and validating the same with reference to the functional requirements. However, no models are presented that help designing the Black Boxes and verification of the same. The survey on embedded systems has recommended the verification and validation to be undertaken based model based checking. These models as such do not support the Black box stricter, which do not consider stimulus response sequences [105] have presented additional verification frameworks to verify and validate the stimulus-response sequences derived through END-TO-END processing and use case models again proving that the stimulus response sequences generated are the same when they are generated using the E2E and Use Case Models. [106] have used the thin thread framework for generating the stimulus-response sequences and compared the same with the Stimulus-response sequences that are generated directly from the requirement specification. The verification and validation of Black Boxes have been presented considering generation of the stimulus sequences from two perspectives.

The new models recommended above have been added to the original CRSE Methodology and the refined CRSE model at Black Box stage is shown in the figure 4. The new E2E Model, Stimulus Precedence Model, E2E Precedence Model and Use case Models proposed shall replace the previous models. The newly recommended models are shown in parallel to the original models. The new Models replace the originally recommended Models.

Refinement at State Box Stage

[99] Has described a methodology for constructing the state boxes with the help of data abstractions. The term "data abstraction" implies the possibility of storing data between stimuli to respond to the effects of previous stimulus.

The principle of information hiding requires that the data that is required as a response at a later operation be regarded as part of an abstract state of the data abstraction, which may be implemented in various ways but always provides a correct description of the behavior. [111] has stated that the state box is the first step of implementing the data specification of black box. In the state box different aspects of implementation of concrete data structures are considered which includes state data, process specification that operates on the data. The abstract models, thus, are restated in terms of the state data. Thus, the state data objects are instances of black box data specifications. The state model exists in real programming sense and the state data have initial values.

[111] have recommended that the state boxes be derived directly from the black boxes by way of implementing abstract data models. Thus, a hierarchy of the black box to a state box is derived and this hierarchy is generally called as usage hierarchy since the relationship between BB and SB is determined by usage of one specification by another. The hierarchy also defines a dependency relationship. When a change is made at implementation level, a re-review at the higher level (BB) must also be undertaken.

[74] Have proposed a method for deriving the State Charts from use cases. They have considered computational requirements of reactive embedded systems as the basis of
representing the functional requirements as a set of state boxes. The computational model of a reactive system must support concurrency and sequential processing and it also must support the communication through different interfaces between the concurrent components that forms the system. The computational model must also support the representation of the system as hierarchical models such as state boxes.

One of the strategies is to implement the state charts using both the direct and indirect implementation [74] have proposed balance approach which includes direct implementation and handling the issues related to communication, hierarchical refinements, concurrency, and parallelism at higher level of the state charts.

Several translation procedures are suggested that lift the issues related to concurrency and communication to the higher levels of the state chart hierarchy. Moreover, the components will be identified such that the implementation of the state charts can be achieved in terms of hardware, software or both. [74] have dealt with the issue of concurrency of execution of various components identified at the state chart diagram level which is a decomposition of the functions that are related to a use case. However, in Co-design of an embedded system, more important issues related to implementation at the hardware, software and hardware-software level and the partition of state chart should use the criteria related to implementation at the hardware, software and hardware-software level. Even the issues such as response, scheduling and reliability must be taken into account so that event based processing can be implemented. Optimization of the performance at the code level both in case of hardware and software must also be addressed and possibly provide the reverse engineering based on the optimized code to represent the application. [99] advocates discovering state data requirements from the responses which defines the necessary information to be maintained in the state box. By appending the state transitions to the state data, the complete state box is obtained. This state box is verified by comparing the intended black box with the derived black box obtained by eliminating the state data.

[111] advocated the usage of representative mapping function that defines a correspondence between elements of the black box data space and elements of the state box data space. The mapping between the state box data space and the black box data space is usually many-to-one and must be onto [112]: for every point in the black box data space, the representation mapping must map at least one point in the state box data space into it. During the definition of the concrete model the state box designer should define the representation mapping. Three correctness questions must be answered in order to conclude that a state box is correct with respect to a black box. Several models have been presented in the literature for designing the embedded systems. Most of the recommended methods are model based. State based models have been used for designing the internal behavior of the embedded systems. Some of the models proposed from the Field of embedded systems and CRSE are state based and therefore there is a straight relevance. However the state models recommended have not been related to Black box structures as recommended in CRSE methodology. CRSE has advocated the Formalism for modeling the internal behavior and also for verifying and validating the same with reference to Block Box structures. However, no models are presented that help formalizing the designing the state Boxes. Sastry JKR et al., [113] have recommended the design of the state boxes by using the UML artifacts. The models not only formalize but also automate the process of generating the state models.

**D Refinement for Verification and validation of State Box Structures**

[111] has clearly made a distinction between specification and design. While the specification is the abstract description of the behavior of the system, the design is towards the implementation of the behavior. A design could be evaluated through assessing correctness and design quality. A design is correct with respect to a specification if the designed behavior and the specified behavior are exactly the same under all circumstances of use. A verification method should be used to verify whether the design is in accordance with expected behavior [99] advocates discovering state data requirements from the responses which defines the necessary information to be maintained in the state box. By appending the state transitions to the state data, the complete state box is obtained. This state box is verified by comparing the intended black box with the derived black box obtained by eliminating the state data.

[111] advocated the usage of representative mapping function that defines a correspondence between elements of the black box data space and elements of the state box data.
space. The mapping between the state box data space and the black box data space is usually many-to-one and must be onto J.D. Gannon, R.G. Hamlet and H.D. Mills [112]: for every point in the black box data space, the representation mapping must map at least one point in the state box data space into it. During the definition of the concrete model the state box designer should define the representation mapping. Three correctness questions are answered in order to conclude that a state box is correct with respect to the Black Box. The Box Description Language (BDL) [98] [100] provides a syntactic description of the three views of a box structure. In order to support the design of an encapsulated box structure, the syntax must be capable of defining an object’s characteristics including different uses of state data to a black box. The review on the ES models reveals that the correctness of the stage models is achieved through model checking. The correctness is verified with reference to integrity, completeness and traceability, However theses models are semi formal and are not automated as such.

Sastry JKR et al., [113] have presented verification models that address the verification and validation of the state models. Verification and validation of the state model has been achieved through building sequence repository and proving that the sequence repository derived from the class diagram is same as the sequence repository derived through E2E modeling. Algorithms have been designed using which various models that formalizes the internal behavior modeling has been achieved.

The new models recommended above have been added to the original CRSE Methodology and the refined CRSE model at State Box stage is shown in the figure 5 The new E2E Model, Stimulus Precedence Model, E2E Precedence Model and Use case Models proposed shall replace the previous models. The newly recommended models are shown in parallel to the original models. The new Models replace the originally recommended Models.

E. Refinement at Clear Box Stage

1) Refinement through generation of code in respect of Both HW and SW

[93] Defines the clear box as the one that opens up the state machine description of a system one more step in an internal view that describes the system processing of the stimulus and state. The processing is described in terms of four possible control structures namely sequence, alternation, iteration and concurrency. Embedded black boxes are placed within these control structures. At this point, a hierarchical, top-down description can be repeated for each of the embedded black boxes at the next level of system description. 98] defines that the clear box of a coherent part is given by a rule for the black box transition function of its state machine, in which the rule is defined in terms of a sequence, alternation, or iteration of other black boxes, or the concurrent effect of other black boxes.

[98] has described a methodology for constructing the clear boxes with the help of data abstractions. There four kinds of clear boxes namely sequence, alternation, iteration and concurrency in which the first three are sequential forms and the last one is the concurrent form. [111] describes the clear box as the final step in implementing a process specification.

A clear box consists of a control structure and lower-level process specifications controlled by that structure. It may also contain local data and it may be defined as a procedure, with a name and parameters. The naming of procedures and the creation of insatiable data specifications are techniques that support the use of common services. Most of the methods discussed above lacks in many of the respects which include inability to generate the code related to model the Hardware and software and also the interfacing between them. Very few models are available that help generating the embedded software. In the literature while some strategies have been explained to generate the embedded software, no specific methodology as such has been recommended as of now. Again no common language is used to model the software and the hardware as a result it becomes necessary to use too many tools and frequent translations leading inefficient methods for generating either the Hardware or Software related code. The performance and scheduling issues are never addressed especially when the embedded software has to run under influence of real time operating system. CRSE has advocated usage of formal models for generating the code. No formal methods as such are proposed in the literature. No research findings have ever been presented to generate code in respect of embedded systems.

[114] have proposed a model for generating code in respect of the Hardware [116] have presented a model for generating code in respect of Embedded Software. The models...
have presented a framework for generating the code on formal foundations. The models address all the basic requirements that must be fulfilled for generating the code. They have been presented models for generation of code related to hardware selection, interaction and layout, and one model related to code generation in respect of ES software and another model for the combined computing of the performance of the hardware and software. The ES software generation has been done considering the internal behavioral models modeled as state diagrams. A novel way of scheduling the ES tasks have presented as part and parcel of code generation for ES software. C++ language has been used as the language for code generation in respect of both hardware and software. The fundamental requirements that must be met by a code generator for generating code for both hardware and software and the integration both have been deliberated and the same are used as the basis of generation of code. A code generation framework as such has been proposed in this chapter.

2) **Refinement through Performance Optimization**

In CRSE no specific models have been recommended that deal with performance optimization. Many performance optimizations issues have been addressed when it comes to embedded systems. The models proposed either addresses the performance optimization either at hardware device level or through Embedded System code optimization level. No models have been recommended that addresses the performance optimization of the embedded systems considering both hardware and software. [115] have proposed a model for undertaking the performance optimization. The models presented for performance optimization considers schedulability, delay times, latency times etc that are the requirements that must be met by the embedded system. The Model presented considers entire system as whole and the Hardware and Software Separately.

3) **Refinement of CRSE at CB stage**

The new models presented for Code Generation in respect of HW and SW and the model related to performance optimization have been added to the original CRSE Methodology and the refined CRSE model at clear Box stage is shown in the figure 6. These models will be added to the original recommendation of CRSE. The newly recommended models are shown in parallel to the original models. The new Models replace the originally recommended Models.

**D. Refinement at Testing Stage**

In CRSE, it has been advocated that the testing and certification of the system be done in a parallel path of the development and the flow commences right from the beginning of the development. The process flow identified by CRSE for testing include the development of Usage models, Markov Chain, Generation of Usage Statistics, generation of test cases based on usage statistics, undertaking the testing using the generated test cases and proceeding towards computation of reliability based on which the certification of the product is undertaken.

1) **Development of usage models**

[123] In CRSE, to facilitate testing of the code generated from Clear Box Phase, a model is developed to characterize the population of uses of the software, and the model is used to generate a statistically correct sample of all uses of the software. The data that can be used throughout the project’s life cycle [124] for test planning, and statistically valid samples of test cases are derived based on the usage model.
modeling the test case generation and also undertaking the test automation. They describe an approach for generating system level test cases based on use case models and refined by state diagrams. They have advocated the transformation of use case models into usage models based on which statistical testing can be carried out. In order to systematically derive the usage models for a software system, the dynamic aspects should be modeled from a usage-oriented point of view, in particular the system’s interaction with the environment.

A well-suited tabular template for textual refining use cases was first published in [146] and has been taken up in [130]. A single, complete description of use case by specifying and mutually relating all of the scenarios it includes is obtained in this textual notation. The proposed templates are also extended through addition of multiple conditions. A scenario step resembling a stimulus will be turned into an atomic test input and a scenario step describing a response will become an atomic observable response. The templates representing the uses cases are enhanced to support the systematic derivation of a state-diagram-based usage specification.

The approach discussed in [130] modified to model each use case into a separate state diagram and is anchored into a top-level diagram. This top-level diagram resembles a framework in which, use cases can be “executed” depending on their pre-conditions, and global usage states which are extrapolated from all of the preconditions of the use cases. The guidelines for constructing the top-level state diagram and individual state diagram for each use case are thoroughly discussed by [130]. The top-level state diagrams are transformed into usage graphs. A detailed discussion on transitions between states is presented [131].

[77] Explained the process of generating automated test cases based on modeling combinatorial dependencies between input parameters and using Markov chain techniques. Input combinations play an important role rather than stimulus sequences in testing of software programs whose usage pattern consists of only three stages, namely, entering input parameters, calculation and generating results [132], [133] suggests different combinatorial approaches for situations where the number of possible parameter values is too large for testing all input combinations. Combinatorics together with Markov chains automates test cases selection, execution and evaluation and allows applying statistical analysis to the testing process. Markov chains are the usage models each path in the usage graph relates to a particular usage of the application.

[135] explains the application of Markov chain model for the selection of test cases from independent input parameters. A specific value represents each state of the model for each discrete input parameter. State transition occurs once the corresponding value is assigned to the parameter. Continuous parameters must be abstracted into discrete parameters by partitioning the parameter’s domain. Probabilities are assigned to determine the usage statistically and random tests are generated automatically for statistical sampling and reliability evaluation. Dependencies between input parameters exist when a value of one parameter imposes some restrictions on the value of the other parameter or affect its exit arc probability distribution. This leads to a set of valid combinations of input parameters and advocates consideration of these dependencies in systematically deriving the models and test case generation.

[77] Have proposed an approach close to the “Conflict-free sub-models” approach of [134], differing in reducing the number of new states and generation of all test cases using only one model. The dependencies are primarily classified into “one-on-one”, “one-on-many”, “many-on-one” and “many-on-many”. The probability that an input parameter has some specific value may depend on the values of other parameters, gives rise to additional dependencies like “Values-on-values” that determines which combination of input parameters are valid or invalid, and “probabilities-on-values” that determines the likelihood of using some input combinations during long run. All the dependencies are used to develop a connected graph which can be used as a usage model. [136] advocated the usage of constrained UML artifacts like use case diagrams, sequence diagrams and the execution probability of each sequence diagram for deriving software usage models. Usage model is generated by projecting the messages in sequence diagrams onto the objects under test and associating probability of occurrence for each message. Testing carried using usage models estimate software reliability.

[137] proposes to combine message sequences from sequence diagrams with category partition testing. [138] adapts the traditional data-flow coverage criteria as test adequacy criteria in the context of UML collaboration diagrams. [139] Derives system test requirements from use cases and sequence diagrams by considering the sequential and dependability relationships between use cases and sequence diagrams. [130] Automatically generates systematic test suites with a given coverage level by transforming use cases into UML state charts and mapping its elements to the STRIPS planning language.

[136] advocates imposing testability constraints on UML artifacts and deriving a Markov chain usage model from them to support statistical testing. [141] Explains the execution sequential relationships of use cases which reflect the business process the system supports. As different use case execution sequences may trigger different failures they must be considered for usage-based software statistical testing. The execution sequential relationships between use cases are represented by activity diagrams [141] with vertices as use cases and edges as execution sequential relations between use cases. Use cases with no relations can execute in parallel. The pre and post conditions will determine the next executed use case. [136] advocated the addition extra constraints to UML artifacts so that the UML models become testable.

The derivation of single use case’s usage model and combining them to get the system usage model is algorithmically explained. The Markov chain usage model thus developed is used for statistical testing, automatic test case generation, measuring software quality, and test adequacy. [142] advocated the usage of operational data such
as users’ activities, log files of the program, etc, for generation of usage distribution which is useful in building the usage model at a low cost. This is achieved by applying source code generation methods based on a state machine diagram by establishing a one-to-one relationship between each transition on the state machine diagram and basic blocks in the skeleton code generated. Source code generation methods addresses the problem of difficulty in deriving the information about state transition of software from its ordinary operational data and also the cost involved in inserting probes into appropriate basic block to gather operational data. It involves automatic translation of state machine diagram into a skeleton program which functions as a part of the software. It supports rapid development and high reliability in the field of embedded systems. Construction of probability distribution of a usage model is done by inserting probes into source codes while generating skeleton codes. These probes collect operational data in the state transition sequences or execution frequencies of each transition. This enables software to collect operational data for building a usage model. Source code generation using state transition table [59] and OMT (Object Modeling Technique) [144] are not yet standardized.

[145] defines the usage model as the one that describes the interactions between the user and the system at a level that identifies the systems’ benefits to the user and presents a structure for usage model that contains three separate tiers: supporting data, overview, and usage details which provide a common taxonomy across various teams and business units, allows reuse of usage model data where appropriate, and aids communication within and between product development teams. Product usage is described by use cases, scenarios, and concept-of-operations documents [147], [148], [149] Proposes use cases to be either translated or extended into concept-of-operations model. A common structure and taxonomy for describing product usage are necessary in order to unify requirements engineering, planning and design processes across business units and promote reuse.

[150] describes a Synthesized Usage Model comprising of actors, usage views, use case specifications, abstract interface objects, user and system actions and a data dictionary. Construction of usage model starts from the beginning of the project. As the model develops supporting data is collected from which usage summary is created. Addition of usage details completes the model. In addition to the coverage of product usage the usage model also contains data about the environment in which the product is used and different types of product users. The scope and depth of usage model development are determined by many factors like experience and domain knowledge of the development team, the precedence of the product within the company, the number of new or risky areas to be addressed, and product complexity. It evolves as the product development progresses and interaction between the strategic planners, user-centered design experts, and engineers occur. The advantages in defining the structure and content of a usage model includes clarifying necessary and intended usage in both new and legacy areas, grounding platform requirements in usage, codifying the role of user-centered design in platform requirements, planning, and development, unifying concepts and vocabulary, and exposing the need for additional work.

[136] advocates using timing constraints in sequence diagrams in generation of usage model for real-time software statistical testing. Timing constrains are expressed by four classes of syntactic constructs [151] namely, Timers to express maximal delay between two events in one process, Delay Intervals to express time intervals between two consecutive events in a process, Drawing rules and timing markers used to express timing constraints. OMG extended UML with a framework for representing time and time related mechanisms OMG [23] to support real-time software development. The timed scenarios can be used to validate timing assignment and verify timing consistency [153]. Sequence diagrams are formalized based on partial ordering of events. A sequence diagram is defined by a finite set of instances, finite set of send and receives events, finite set of messages and a set of timing constrains. A Use case is defined by a set of sequence diagrams associated with the use case and set of preconditions of the use case specified with Object Constraint Language (OCL) [154]. By adding execution post condition and execution probability of the sequence in its associated use case to the sequence diagram we get a sequence diagram with statistical testing constrains. A Markov chain usage model is generated for these sequence diagrams assuming that the objects under testing are totally ordered. The usage model for a use case is generated by integrating the usage model of all the sequence diagrams of the use case. An algorithm to generate Software usage model is presented. Statistical test cases for testing real-time systems are automatically generated by taking timing constrains also into account.

[143] advocates the usage of high-order Markov chains for constructing accurate usage models. Statistical testing overcomes the shortcomings of the systematic testing in expressing software reliability Whittaker [127], [124], [123]. The effectiveness of the statistical testing is determined by the accuracy of the usage model. By using high-order Markov chain in which the immediate past state is also considered along with the current state in determining the probability of event occurrence the accuracy of the usage model is increased. Constructing an accurate usage model involves gathering the reliable field data and constructing the detailed usage structure. It requires a method of creating equivalent states at realistic cost. Using high-order Markov chain supports automation and the object of creating an equivalent state is not limited to a recurrent state. [90] advocated the usage of sequence-based requirements specification in combination with model-based statistical testing for very high degree of automation from requirements document to statistical test report. Sequence-based specification is a set of techniques for stepwise construction of black box and state box specifications of software systems. It helps in analyzing the completeness and consistency of the requirements with a stepwise construction of a traceably correct black box specification. [125]

2) Model Based Testing
UML based statistical testing requires, acquisition of execution probability of the messages in the sequence diagram associated ith use case and projecting them onto the objects of the system under test to generate usage models. Preparation of system usage model by integrating the usage models of the use cases and testing the system with the test cases generated from the software usage model to estimate software reliability is presented in [125]. In statistical testing of software, all possible uses of the software, at some level of abstraction, are represented by a statistical model wherein each possible use of the software has an associated probability of occurrence [128]. Test cases are drawn from the sample population of possible uses according to the sample distribution and run against the software under test. Various statistics of interest, such as the estimated failure rate and mean time to failure of the software are computed. The testing performed is evaluated relative to the population of uses to determine whether or not to stop testing.

3) Markov Chain usage models

Markov chains are stochastic models [155]. A specific class of Markov chains, the discrete-parameter, finite-state, time-homogenous, irreducible Markov chain, has been used to model the usage of software. The Markov approach to usage modeling provides procedures to make the testing process more efficient. [127], describe procedures for using analytical results known for Markov processes to provide insight into testing and for using Markov chains to generate random test cases, conduct statistical tests, and determine stopping criteria. In a Markov chain usage model, the states of use of the software are represented as states in the Markov chain [123], [127], [124]. User actions are represented as state transitions in the Markov chain. The probability of a user performing a certain action, given that the software is in a particular state of use, is represented by the associated transition probability in the Markov chain. Given a Markov chain based usage model, it is possible to analytically compute a number of statistics useful for validation of the model, test planning, test monitoring, and evaluation of the software under test. Example statistics include the expected test case length and associated variance, the probability of a state or arc appearing in a test case, the long run probability of the software being in a certain state of use [124].

4) Test Case generation and Conducting testing

Test cases are randomly generated from the usage model, i.e., randomly sampled based on the use distribution. These test cases are run against the software and estimates of reliability are computed. The state diagram of an mth order Markov chain is defined as “detailed usage structure” where m-1 represents the number of past state transitions stored in each state of the detailed usage structure. It differs from the “normal usage structure” by having equivalent states. Event sequences as field data are gathered and are mapped to normal and detailed usage structure. Probability distributions are calculated by tracing the detailed usage structure thereby developing the usage model of the mth-order Markov chain. A formal definition of normal/detailed usage structure and of event sequences is presented. High-order Markov chain generates test cases that the normal usage structure can’t reveal. It increases the accuracy of evaluation of software reliability and also increases the effectiveness of statistical testing. [143] advocates not using an extremely high-order Markov chain as humans lose their past memory and usage characteristics may result from their recent operations rather than their distant past ones. The suitable order for each development situation can be found by setting some pre-defined conditions on properties like entropy, number of states, number of transitions etc or by extracting events having significant dependence on usage characteristics. The cost of the construction of the usage model is determined by its construction time and the order with a high cost/performance ratio can be determined through experiments.

[133] suggests different combinatorial approaches for situations where the number of possible parameter values is too large for testing all input combinations. Combinatorics together with Markov chains automates test cases selection, execution and evaluation and allows applying statistical analysis to the testing process. Markov chains are the usage models each path in the usage graph relates to a particular usage of the application. [77] explained the process of generating automated test cases based on modeling combinatorial dependencies between input parameters and using Markov chain techniques. CRSE has not proposed any specific method for undertaking actual testing and conducting the audit trail though references to tools like JUMBL Prowell S.J. [156] have been made using which the actual software testing can be undertaken.

5) Certification through reliability

CRSE advocated certification be done through reliability assessment of the systems. Several reliability estimation models have been proposed that help computing the reliability of a Computing System. Reliability can be evaluated either through Formula based or model based. Formula based Reliability evaluation is undertaken either trough the Miller [159],[160]. The model based reliability computation is quite applicable for computation of reliability when models are used for testing [158], [161]. The effectiveness of each of the model must be studied as the embedded system involves both hardware and software and an interface between them. It is necessary to compute the software reliability, hardware reliability and the combined reliability. The model based reliability computations as such ignores the failures due to software. Computing the reliability is not possible when testing reveals no failures. Alternative strategy to estimate the reliability is required, [157] has proposed a method to compute the reliability when testing reveals no failure.

6) Refinement of CRSE for facilitating Testing of Embedded Systems

Several issues must be considered when comprehensive testing of the embedded systems is needed which include Location of Testing, Methods sued for testing, gadgets used for testing etc. The Testing Path defined in the Clean Room Software Engineering Methodology for undertaking testing and certification uses various types of models that include Usage models, Probability Distribution models, Markov chains, Generation of statistics that best defines the usage
models, Generation of test cases based on Generated Statistics and certification of the product through Reliability Models. CRSE however have not defined any method for undertaking the actual Testing. The Testing and Certification models recommended by CRSE are primarily dependent on building models based on the USAGE specification that is drawn from the Application requirements. The way the system is expected to be used is the primary basis on which the testing is undertaken.

Embedded systems on the other hand must be tested from the point of Event processing, Event Pattern processing, testing for response time, Testing for Through Put and testing for Criticality. None has proposed any method of a model that incorporates all these issues and that facilitates test case generation and undertaking the testing. The code is generated after completing the design of clear boxes. The code generated for HW is simply related to the selection, integration and display of the Hardware and the code is not used for testing the proper running of the Hardware. The code generated related to embedded software must be tested thoroughly, in addition, the hardware related to embedded system must also be tested by using some other method. The interfacing of Hardware with software must also be tested.

CRSE has advocated the testing and certification flow keeping in view of the software alone but when it comes to embedded systems, the testing of hardware, embedded software and the integration of hardware with software should be undertaken. The testing flow recommended by CRSE is completely based on specification that describes how the system is going to be used. This is completely a different view of the requirements specification. But when it comes to the embedded systems, the issues related to event processing, response, throughput, sequence of occurrence of events, criticality of the systems, signal flow, signal balancing etc must be tested.

The CRSE recommended testing and certification method is useful only for the loaded systems but not for Real-time embedded systems. The testing of embedded system involves testing the hardware, software and both. The methods and the locations used for undertaking the testing of embedded systems are quite different from the methods used for testing the loaded systems as defined by CRSE.

[89] has recommended models using which the embedded systems can be tested comprehensively. The models suggested by them considers all aspects of testing which includes testing of hardware, software and both using different methods such as scaffolding, Assert Macros, third party tools, In circuit emulator, instruction set simulators, Third party tools, Logic Analyzers and Monitors. The embedded systems can be comprehensively tested using the models suggested by them. The code that is generated at the stage of clear box design can be subjected to the models recommended by [89] and the embedded system is tested by generating test cases of different types suitable for testing using different test methods.

The test cases and test results obtained by applying the models can be used for estimating the reliability of the embedded systems from multiple angles. The testing and certification path is modified with the models recommended for arriving at the overall Refined Clean Room Software Engineering methodology that best suits to manage the life cycle of the embedded systems. The test cases and the test results are presented and the reliability of the embedded system is computed and found to be 98% reliable.

VI THE COMPLETE PROPOSED REFINEMENT FOR THE CRSE MODEL

The completely refined CRSE model has been presented in the Figure 7. The completely refined CRSE methodology includes all the models that are situated in both the paths of CRSE Methodology. All the models presented provides for formal structures and complete verifiability and validation especially in the development path of the refined CRSE methodology.

VII CONCLUSIONS

CRSE has been recommended with the main aim of delivering high quality systems based on formal models and complete verification and validation at every stage of development. The models included in CRSE methodology also aim at minimal testing. CRSE aims at developing high. However most of the models available in literature that are used at every stage of CRSE methodology are either manual, semiformal or formal with high level of complexity

REFERENCES

Fig 7 The completely refined CRSE methodology

[9] E.Gamma, R.Helm, R.Johnson and J.V lissides, 1994], Design Patterns: Elements of Reusable Object oriented Software, Addison-Wesley


[18] Petri Kukkala, Jouni Riihimaki, Markov Hannikainen, Timo D.Hannikainen and Klaus Kronlof, 2005, UML 2.0 profile for Embedded system design, Proceedings of the design, automation and test in Europe conference and exhibition DATE’05, March 11, 18


[22] OMG, 2003], Response to the OMG RFP for schedulability, Performance, and Time, OMG Document Number: ad/2001-06-14, June


[34] Sergio Gavaldà-Rodes, E.G. Sirer, and D. Tse, 2003, A component-based methodology for Embedded system prototyping, Proceedings of the 14th IEEE International work shop on Rapid systems prototyping, March, 9, 15


[56] Fagan, 1976, Design and code inspections to reduce errors in program development, Reprinted from IBM Systems Journal, VOL15 NO 3


[69] Matteo Bordini and Tulio Vardanega, 2007, Real-Time Java from an atomated code generation perspective, JTRES’07 Proceedings of the 5th international workshop on Java technologies for real-time and embedded systems, sept, 26, 88


[74] Min Zhao, Bruce Childers and Mary Lou Soffa, 2003, Predicting the Impact of Optimizations for Embedded systems, LCTES’03 ,ACM, June, 11, 13


[83] Yong-Yoon Cho, Jong-Bae Moon and Young-Chul Kim, 2005, A system for performance Evaluation of embedded Software, World academy of science, engineering and technology


[147] Cockburn A., 2001, Writing Effective Use Cases, Addison-Wesley

[148] Fairley, R.E.,Thayer, R.H., and Bjorke, P., 1994, The concept of operations: The bridge from operational requirements to technical specifications, Proceedings of First International Conference on Requirements Engineering, April, 40, 47


