International Journal of Embedded Systems, Robotics and Computer Engineering.
Volume 1, Number 1 (2015), pp. 7-12

http://www.irphouse.com

A Study on Issues Related to Software Dependability for Critical Embedded Systems

Ranee Lilhare, Nidhi Choubey and Anuj Kumar Dwivedi

School of IT, MATS University, Raipur, C.G., (India)-493447
raneelilhare@gmail.com, nidhi5539@gmail.com
anuj.ku.dwivedi@gmail.com

Abstract

Today, embedded systems are used in almost every critical applications. The ultimate goal of an embedded system is to achieve high level of quality and dependability. The goal of this research study is to investigate the inter-relationships between dependability and other embedded systems quality attributes using two pieces of information: Tactics and dependability Quality attributes scenarios.

Keywords: Software dependability, Software engineering, software reliability.

1. Introduction

As electronic grown, people's dependency increases on embedded devices. The area of applications of these embedded devices increases day by day. These embedded devices are controlled by some form of tiny operating systems specifically designed for controlling these new generations of devices controlling critical applications. Dependability can be defined as the property of a system to deliver justifiable service and to avoid failures that are serious and numerous. Dependability is very important for embedded systems. Quality attributes is a superset of Dependability. Dependability quality attributes defines what the acceptable behavior should be in case of attacks, faults, mishaps and failures that occur when the system is operating, and how much effort required for the modifications needed to correct errors.[4][6]

Modern systems are critically dependent on software for their operation and design. The next generation of developers must be too easy in the design specification, and implementation of dependable software using accurate developmental processes.

As software in an embedded system is being used of controlling both software and hardware components, Software reliability is a important factor of an embedded

system that increase the performance of a computer. software quality must be measured in terms of results, not in terms of effort expended, however well meant.

Today, computer has become very crucial to perform critical tasks in which failure can have very high costs, we all know computer guarantees that programs will perform properly. High quality of software is responsible for reliability of computer. Dependability is most important part of embedded system .dependability means reliability, safety, security, these features are available in software then computer performance is increased.

This joined with increasing software complexities and large software size requires a very different approach in assessing software reliability in the form of reliability or "dependability" can be demonstrated by the successful completion of testing that "covers" the software. From such computers, High level dependability is required, including their software. For software used in the most critical roles, such visualizations are not usually supplied.[1]

2. Importance of Dependability

Dependability becomes more important due to the following factor:

- Systems that are unsafe, unreliable or insecure and not dependable may be rejected by their users.
- System failure may have very high cost.
- Reliability only covers following factors: *security*, *correctness*, *robustness*. Undependable systems may cause information loss with a high subsequent recovery cost.

2.1 Correctness, Robustness, Security

We may rely on the following definitions for these three factors:

- The ability of a system to execute according to its specification in cases of use within that specification is Correctness.
- The ability of a system to prevent damage in case of erroneous use outside of its specification is Robustness
- The ability of a system to prevent damage in case of hostile use outside of its specification is security.

Quality of software must be measured in terms of results, not in terms of effort expended. Unreliability of any product comes due to the presence of faults or failures in the system. Software does not exhaust or age, as an electronic system or a mechanical system does, the unreliability of software is primarily due to bugs in the software or design faults. Reliability is a measure of probability that assumes that the occurrence of failure of software is a random phenomenon.

3. Scope of Dependability

- Availability
  - when requested the ability of the system to deliver services.

  - Dependable means available with respect to the readiness for usage.
- Reliability
  - Dependable means reliable with respect to the continuity of service.
- Safety
  - Dependable means safe with respect to the avoidance of terrible cost on the environment.
- Security
  - Dependable means Secure with respect to the prevention of unauthorized access and/or handling of information.

Availability is the readiness of service for authorized users. It is the measurement of duration it would take an intruder to cause a denial of service. This attribute focuses on the system behavior versus the faults encountered during the system operation.[1][5]

Reliability is the continuity of service. The system is expected to execute its task in spite of the existence of some faults. This quality attribute is concerned with demonstrating acceptable behavior of the system when faults are encountered.[5]

Integrity is the non-occurrence of improper alternation of information. In case of attacks, this quality attribute is concerned with system behavior.[6]

Confidentiality is the non-occurrence of unauthorized disclosure of information as system data and programs are resistant to unauthorized modifications.

In case of attacks, this quality attribute describes how the system will behave in case of attacks. Safety is the non-occurrence of catastrophic consequences for the user(s) and in the operation environment. How the system should deal with mishaps and/or failures when thee occur is described by this quality attribute[6].

4. Software Reliability Activities

The reliability process is a model of the reliability-oriented aspects of software development, operations and maintenance. Project reliability profile include artifacts, errors, defects, corrections, test, faults, failures, outages, repairs, validation and expenditures of resources such as CPU time, manpower effort and schedule time. [6] The activities are grouped into classes:

- Construction
  Generates code artifacts and new documentation
- Combination
  It integrates code components and reusable documentation with new documentation and code components.
- Correction
  Defects are analyzed and removed in documentation and code using static analysis of artifacts.
- Preparation
  Generates test cases and test plans, and readies them for execution.
- Testing
  Executes test cases where failure occur

- Identification
  Makes fault category assignment. Each fault may be new or existing
- Repair
  Removes faults
- Retest
  Executes test cases to validate whether specified repairs are complete or not.

5. Deficiencies

In trying to ascertain the reliability of a software product or process we must adopt a negative mindset and look for sources of violation of reliability properties.[5] The terminology distinguishes three levels:

- A failure causes degradation in performance.
- A fault is a departure of the software product from the properties it should have satisfied. A failure always comes from a fault. A fault could be in the specification, in the documentation, or in a non-software product such as the hardware on which the system runs.
- An error is a wrong human decision made during construction.

6. Software Reliability Improvement Techniques

Good engineering can largely improve software reliability. In real case, it is not possible to eliminate all the bugs from software; however, by applying good software engineering principles software reliability can be improved to a great extent. The application of disciplined, quantifiable approach to the development operation and maintenance of software will produce economic software that is reliable and works efficiently on real machines.[6]

6.1. Process

Process defines a framework that must be establish delivered software technology. It is the basis for management control of software projects and establishes the context in which technical methods are applied.

The process itself must be assessed to ensure that it meets the basic process criteria that are necessary for successful software engineering

6.2 Software Engineering Methods

These methods consist of an array of tasks that include requirement analysis, design modeling, program construction, testing and support.

6.2.1. Requirement Analysis

In early days, all the focus was on coding and testing, but researchers have shown that requirement analysis is the most difficult and intractable activity and is very error prone. In this phase software failure rate can be increased by:

- Properly identifying the requirements
- Specifying the requirements in the form of SRS document.

- Validating the SRS.
- Developing Prototypes.
- Performing structured analysis for developing conceptual models using DFDs
- Make estimations of effort, cost and task duration.
- Performing the Risk management which involves risk management and control

6.2.2. Modeling Design

Design is the first step in moving from problem domain to solution domain. The goal is to product the model of the system which can be later used to build up the system. In this phase, reliability can be improved by [10]:

- Using "Divide & Conquer" method
- Abstraction of components so that maintenance will become easy
- Performing different levels of factoring
- Controlling and understanding the interdependency amongh the modules
- Design reviews to ensure that design satisfies the requirements and is of good quality.
- Developing design iteratively

6.2.3. Program Construction

It includes coding and testing. In this phase, software reliability can be increased by[12]:

- Constraining algorithms by following structured programming
- Write self-documenting code.
- Creating interfaces that are consistent with architecture,
- Conducting a code walkthrough.
- Performing unit tests.
- Refactoring code.

6.2.4. Testing

After coding, testing, verification and validation are necessary steps to follow. Software testing is used to trigger, locate and remove software defects. Various analysis tools such as fault tree analysis, trend analysis, orthogonal defect classification and formal methods etc, can be used to minimize the possibility of defect occurrence after release and therefore improve software reliability.[7][8]

- Verification is *internal* estimation of the consistency of the product, considered just by itself. The last two examples illustrated properties that are subject to verification: for code; for documentation.
- Validation is *relative* estimation of a product vis-à-vis another that defines some of the properties that it should satisfy: code against design, design against specification, specification against requirements, documentation against standards, observed practices against company rules, delivery dates against project milestones, observed defect rates against defined goals, test suites against coverage metrics.

Conclusion

Computers are playing very important role in our life and there is always a need of high quality software. Te most measurable aspect of software quality is software reliability. Unlike hardware, software does not age, wear out r rust, unreliability is mainly due to bugs or design faults in the software. The exact value of product reliability is never precisely known at any point in its lifetime. The study of software reliability can be categorized into three parts: Modeling, Measurement and Improvement. There are many models available, but no single model can acuter a necessary amount of software characteristics. Software reliability measurement is naive. It cannot be directly measured, so other related factors are measured to estimate software reliability. Software reliability is necessary and hard to achieve. It can be improved by sufficient understanding of software reliability, characteristics of software and good software design. Complete testing of software is not possible; however sufficient testing and proper maintenance will improve software reliability to great extent.

References

[1] Avizienis, J. Laprie, and B. Randell, 2000, "Fundamental Concepts of Dependability," Proc. 3rd Information Survivability Workshop, Boston, Massachusetts, USA, pp.7–12.

[2] Ghezzi, C., Jazayeri, M., and Mandrioli, D., 2003, Software Engineering, 2nd edition, Prentice Hall.

[3] The IEEE Technical Committee on Dependable Computing and Fault Tolerance-IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance. Dependable Computing and Fault Tolerance, Available at: www.dependability.org.

[4] Jackson, M., 2001, Problem Frames: Analysing and Structuring Software Development Problems, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, ISBN:0-201-59627-X.

[5] Meyer, B., September 2006, Dependable Systems: Software, Computing, Networks, eds. Jürg Kohlas, Bertrand Meyer, André Schiper, Lecture Notes in Computer Science 4028, Springer-Verlag, Germany.

[6] Quyoum, A., Dar, M.Ud-Din, Di, M.K.Q. November 2010, "Improving Software Reliability using Software Engineering," International Journal of Computer Applications, 10(5), ISSN: 0975–8887.

[7] Binder, R., 1999, Testing Object-Oriented Systems: Models, Patterns, and Tools, Addison-Wesley, ISBN 0-201-80938-9.

[8] Reid, S., 2005, The Art of Software Testing, 2nd edition, Glenford J. Myers. Revised and updated by Badgett, T., Thomas, T. M, and Sandler, C., John Wiley and Sons, New Jersey, U.S.A., 2004. ISBN: 0-471-46912-2, pp 234, Software Testing, Verification and Reliability, 15 (2), pp. 136–137, June 2005.