International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Certification of Components and Component Based Systems

Vijay[1], Devender Kumar[2]

Department of Computer Science & Engineering, Meri College of Engineering & Technology
Sampla, Bhadurgarh, Haryana, India

Assistant Professor, Dept. of Computer Science & Engineering, MERI College of Engineering & Technology
Sampla, Bhadurgarh, Haryana, India

Abstract: ***Component Based Software Development (CBSD) is focused on assembling existing components to build a software system, with a potential benefit of delivering quality systems by using quality components. The quality of a component-based system depends on the quality of its components, and a framework and integration process used. Hence, techniques and methods for quality assurance and assessment of a component-based system would be different from those of the traditional software engineering methodology. It is essential to quantify factors that contribute to the overall quality, for instances, the trade off between costs and quality of a component, analytical techniques and formal methods, and quality attribute definitions and measurements. In this work I explored about Component certification, Component certification techniques, IEEE 1517 standard, Component testing technologies like black box testing, Software fault injection and operational system testing.***

Keywords: API, Applications, Black Box Testing, IEEE1517, OTS, Software Fault Injection, Various approaches.

1. Introduction

Software, as a product, delivers the computing potential embodied by computer hardware. And it is an information transformer producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia simulation [1].

Software engineering is defined as the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software. It is based on three layers, that is, process, methods, and tools. Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Software engineering methods provide the technical way for building software. Methods encompass a broad array of tasks that include requirements analysis, design, implementation, testing, and maintenance. And finally, software engineering tools provide automated or semi-automated support for the process and the methods [1].

According to Pressman [1], Software engineering moved into its fourth decade in the 90s. In the 70's the first traditional methodologies were defined. It is meant that software development using a set of mature and stable technologies, which often include mainframe-based technologies, structured analysis and development techniques, and procedural languages such as COBOL and RPG. Applications often, built using this approach, are used on mainframes and minis. Starting in the 80's, the theory and practice of the object-oriented approach evolved. This usually means adopting an object-oriented methodology based on object-oriented languages. During the 80s, the object-oriented approach was expanded to a theory that covered most of the aspects of software development, including testing and project management. Now a days , component-based development (CBD) is in the leading edge phase. Indeed, there are now a number of technologies appropriate for, and people with experience in, the application of CBD. The evolution of software industry from 1970's to 2014 has evolved with different approaches as structured, object oriented, distributed systems and component-based approach. In industry there is of course a history of this maturity as illustrated in Figure 1.1 and described below [2].

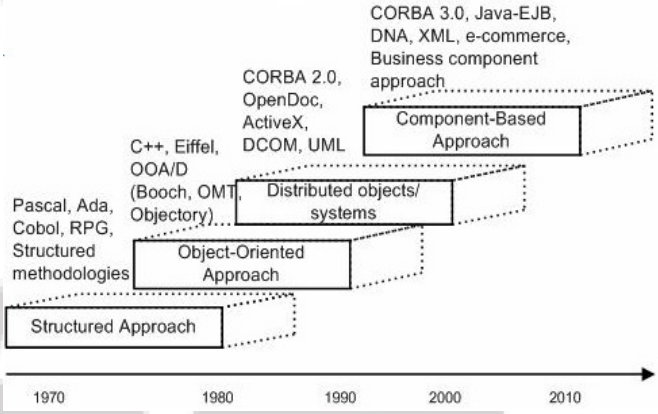

Figure 1: The evolution to components in the industry

The objective of the certification is evaluate the component's quality and component based system process by using standards such as IEEE 1517 and testing techniques like Black Box Testing, software fault injection, operational testing.

2. Methodology

This paper selects one of the Standard IEEE 1517 for assurance of components and component-based systems. Since it is very difficult to certify or assure the quality of components from various vendors and ever changing requirements of software industry. Therefore, this study redevelops it through applying technologies for certifying

component and component-based system by using black box testing, fault injection system and operational system testing.

3. Prior Approaches

- **Structured Approach:-** This approach contains basic steps of a software development process such as analysis, design, implementation, testing, and maintenance. Each of the elements of the analysis model [1] provides information that is required to create a design model. The flow of information during software design is illustrated in Figure 2 as below.

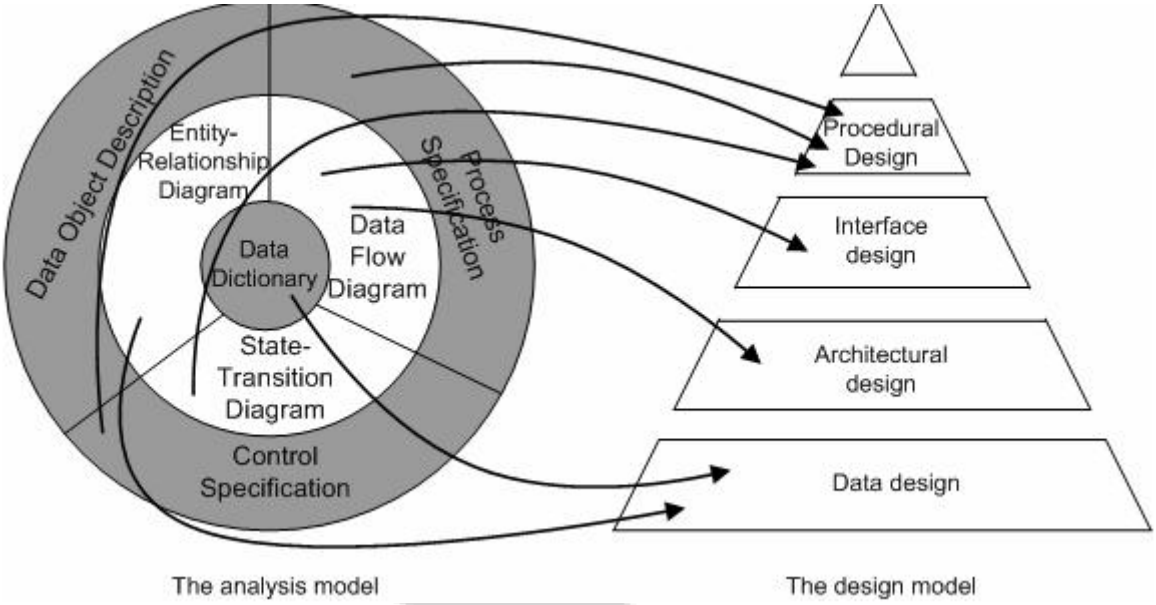

Figure 2: Analysis and design models for structured approach

For specifying software, this approach offers some variety of elements such as a data dictionary, data flow diagrams, state transition diagrams, entity-relationship diagrams; process specifications, control specifications, and data object descriptions for analysis model. The design phase produces a data design, an architectural design, an interface design, and a procedural design with the help of various methods and techniques. A data flow diagram consists of processes, data flows, actors, and data stores. The data dictionary contains details missing from data flow diagrams. Entity-relationship (ER) diagrams highlight relationships between data stores that otherwise would only be seen in the process specifications. Each ER data element corresponds to one data flow diagram data store.

In design phase, the most favorite technique is structured programming to produce procedural design. Languages such as Pascal, Ada and C perform it. The broad definition of structured programming refers to any software development technique that includes structured design and results in the development of a structured program. Structured programming allows programs to be broken down into blocks or procedures, which can be written without detailed knowledge of the inner workings of other blocks, thus allowing a top-down design approach or stepwise refinement [3].

- **Object Oriented Approach:-** Object-oriented approach promises a way for implementing real-world problems to abstractions from which software can be developed effectively. Object-orientation offers conceptual structures that support this sub-division. Object-orientation also aims to provide a mechanism to support the reuse of program code, design, and analysis models [5]. A class is 'A description of a set of objects that share the same attributes, operations, methods, relationships and semantics.' An object is 'an instance that originates from a class, it is structured and behaves according to its class.' In object-orientation, three main principles are important. Encapsulation, which is also known as information hiding, provides the internal implementation of the object without requiring any change to the application that uses it. The ability of one class of objects to inherit some of its properties or methods from an ancestor class is named inheritance in object technology. Polymorphism is producing various results for a generalized request based on the object that is sent to. The object-oriented approach allows development-time reuse [2], meaning that compared to previous approaches, it enhances developers' ability to build software that reuses pieces designed and coded by other developers.

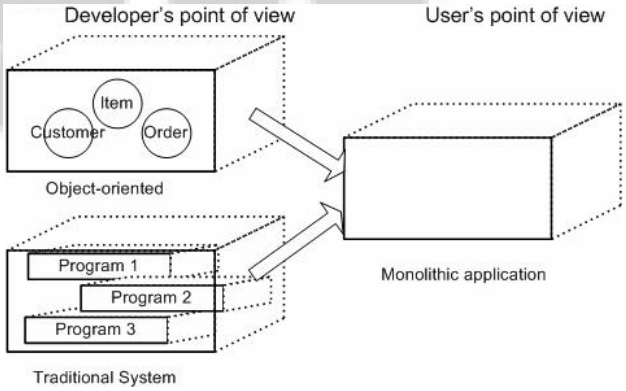

Figure 3: Object-oriented application

4. Our Approach

Component Certification is defined as "To attest as certain; give reliable information of; confirm, to testify to or vouch for in writing, to guarantee; endorse reliably; to certify a document with an official seal." Component certification [4] is the process that involves:

- Component outsourcing: managing a component outsourcing contract and auditing the contractor performance;
- Component selection: selecting the right components in accordance to the requirement for both functionality and reliability; and
- Component testing: confirm the component satisfies the requirement with acceptable quality and reliability
- The objectives of component certification are to outsource, select and test the candidate components and check whether they satisfy the system requirement with high quality and reliability.

3.1 Technologies

- **Black Box Testing:** Black box testing is a family of software testing techniques that selects test cases without regard for the software's syntax. To perform black box testing, an executable component, inputs, and an oracle (which decides if failure has occurred) are needed. In contrast, white-box testing techniques consider the code when selecting test cases. For example, one white-box testing strategy to selects test cases such that each possible outcome from each decision point occurs. Black box testing is not without criticism, however. Black box testing can fail exercise significant portions of the code, and from a certification perspective, that is worrisome. The value-added by black box testing is also dependent on having accurate oracles. Faulty oracles are capable of allowing bad software to be certified. Faulty oracles can also allow quality software to not get certified Correct oracles are of paramount importance to our certification methodology and correct oracles may be difficult for a component purchaser to derive. Our recommendation here is that the OTS consumer develop their own oracle according to what they want the component to do not necessarily according to the specification from the OTS vendor. By doing this the consumer can test the quality of the component against their requirements for what the component is expected to do and not necessarily against the functionality claimed by the vendor[20]. There is yet another serious problem with OTS software and that is that OTS components can have unknown malicious functionality In short blackbox testing plays an important yet limited the quality of software components Admittedly the costs of performing blackbox testing by a OTS consumer cannot be overlooked.
- **Software Fault Injection:-** There are many different forms of fault injection. The particular fault injection technique that our certication methodology will use is called Interface Propagation Analysis (IPA) [6] perturbs i.e. corrupts that propagate through interfaces between components to perturb states access to the interfaces that components use for communication is needed. We use a small software routine to a different corrupt software executes called a perturbation function that actually makes the replacement from the original state. To observe the system wide impact of component failures IPA must be told what component failure modes to inject and what system failure modes to be on the lookout for IPA then checks to see if any of these system failure modes manifest System failure modes can be defined as faulty system output data faulty global system data or corrupted data owing between successor components Since the OTS consumer is the system builder it is reasonable to expect that they know what constitutes as a system failure.
- **Operational System Testing:-** In addition to system level fault injection operational system testing with a OTS component embedded is a complementary means for determining how the system will tolerate the component Operational system testing executes the full system with system test cases. The difference between operational system testing and fault injection is that operational system testing does not employ perturbation functions to perturb states. The output information a component produces does not get modified and the system is executed using original states. An advantage here over fault injection is that when a component really fails the system experiences an actual component failure. This provides a more accurate assessment of system tolerance. The downside is that if the component rarely fails an enormous amount of system level testing will be necessary to make that determination Operational system testing will however be valuable for system level reliability prediction.

Assurance: Assurance is the major issue for quality of software used in the world. The business of global reuse of components via the World Wide Web cannot succeed unless the quality of components cannot be assured.

IEEE Std. 1517: IEEE 1517 was published in 1999 as a supplement to IEEE/EIA 12207 (IEEE, 1999). It is intended as a specification of the minimum requirements that a software life cycle model must meet to enable and support proper reuse.

3.2 Application of IEEE Std. 1517

IEEE Std. 1517 applies to:

- The acquisition of software and software services
- The acquisition of assets
- The supply, development, operation, and maintenance of software applications and systems using a CBD approach
- The supply, development, management, and maintenance of assets
- The establishment of a systematic reuse program and components strategy at the organization or enterprise level

3.3 Static Analysis for Assurance of Component – Based Software

Static component analysis that extracts and records relevant information from previously developed components is a tractable approach for improving assurance and offers some advantages over dynamic analysis.

5. Conclusion

The conclusion of the paper is focusing on development of component assurance taxonomy, as well as information-extraction and analysis strategies. I have identified assurance in component-based software, and approaches supporting analysis of these properties. The IEEE 1517 standard may be better one to get assured about the quality. A first step is the identification of analysis techniques and approaches that can be used in the determination of these properties. I also wrote about static information extraction techniques that can be applied to component-based software. Where applicable, these strategies will be codified in an analysis workbench specification, a prototype analysis workbench, and in criteria useful for the development, acquisition, and analysis of component-based software. Static analysis that extracts and records relevant information from previously developed components is a tractable approach for improving assurance and offers some advantages over dynamic analysis.

Reference

[1] Pressman Roger S, Software Engineering, Tata McGraw Hill, 2006.
[2] Herzum Peter, Sims Oliver, Business Component Factory, Wiley, 1999
[3] Huizing M., Component Based Development, 2000
[4] Cai Xia, LYU Michael R., wong Kam-Fai, KO Roy, Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes.
[5] Hill, Bennett, McROBB, Farmer, Object Oriented System Analysis and Design (using UML), 2nd Edition, McGraw Hill, 2002
[6] Jerey Voas, An Approach to Certifying Off-the-Shelf Software Components Reliable Software Technologies Corporation.

Author Profile

Vijay has done B.Tech in Information Technology (Session 2008-2012) from MERI College of Engineering, Asanda (Bahadurgarh) affiliated to M.D. University, Rohtak (Haryana). He is pursuing M.Tech in Computer Science & Engineering (Session 2012-2014) from MERI College of Engineering, Asanda (Bahadurgarh), India.