8-15-1997

Implications of Using Cleanroom Software Engineering to Develop Highly Reliable Three-Tier Client/Server Applications

Peter Grillo
Temple University, pgrillo@thunder.ocis.temple.edu

Rajiv Tewari
Temple University, tewari@trek.cis.temple.edu

Follow this and additional works at: http://aisel.aisnet.org/amcis1997

Recommended Citation
http://aisel.aisnet.org/amcis1997/144

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 1997 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
Implications of Using Cleanroom Software Engineering to Develop Highly Reliable Three-Tier Client/Server Applications

Peter Grillo and Rajiv Tewari
Temple University
Division of Computer and Information Science
Philadelphia, PA 19122
pgrillo@thunder.ocis.temple.edu
tewari@trek.cis.temple.edu

Research Motivation and Background

The quality and reliability of computer software are critical for an organization's survival and success [Sanders and Curran]. In the past, a software product's functionality was a key to success. With today's increasing global market, competitors can match software functionality quickly and easily. The quality and reliability of the software product in the market and the quality of support that accompanies it will differentiate an organization [Sanders and Curran].

Software development is a process in which a developer translates requirements into software products. The quality of the software is dependent on the quality of the process used [Humphrey; Sanders and Curran; Linger]. Improving the quality of the process used will allow organizations to deliver higher quality products consistently and economically [Sander and Curran]. The need for high quality software and for discipline in the software development process motivates process improvement methodologies [Mills, Dyer and Linger].

One of the most influential advocates of the need to focus on process quality in the software industry is the Software Engineering Institute (SEI). The SEI has developed a five step road map for organizations to follow in improving the software process called the Capability Maturity Model (CMM) [Paulk, et al.]. The core principle of the CMM is that the quality of the software product depends on the quality of the software process used to create it. To consistently improve software products, the process used for development should be understood, defined, measured and improved [Paulk, et al.]. Gibson [1997] suggest the use of Cleanroom Software Engineering (CSE) as an approach for organizations that are trying to achieve higher CMM levels.

CSE is a theory-based, team oriented approach to developing and certifying highly reliable software systems under statistical quality control [Mills, Dryer and Linger; Dyer; Linger]. CSE treats software development as an engineering process based on a mathematical foundation, rather than as a craft-based trial-and-error programming process [Linger, Mills and Witt]. The main objective of CSE is to develop a software product that has a high probability of zero defects and a high measure of operational reliability [Linger].

CSE emphasis is on preventing errors from the beginning rather than removing them later [Mills, Dyer and Linger]. This is accomplished by integrating formal methods for specification and design, non-execution-based program development and statistically based independent testing. The basic components of the CSE process are [Mills, Dyer and Linger; Dyer; Linger]:

1. **Incremental development process model.** Instead of considering design, implementation, and testing as sequential stages in a software life cycle, incremental software development consists of a sequence of product increments. Each increment is developed using the Cleanroom process. These increments accumulate over the development life cycle with the final increment being the product with full functionality.
1. **Formal Methods for Specification and Design**: Cleanroom developers use structured specifications to divide the product functionality into deeply nested subsets that can be developed incrementally. Cleanroom views programs as rules for mathematical functions where programs carry out transformations from input (domain) to output (range) [Linger]. Transformations are specified as functional mappings [Mills, Linger, and Hevner; Dyer; Linger] using the Box Structure Method [Mills, Linger and Hevner]. Box structures provide a stepwise refinement and verification process based on three levels of abstraction for describing programs as rules: black box, state box and clear box forms. The black box is considered the specification and the state box and clear box are considered design.

2. **Correctness verification of developed code**: In Cleanroom the development team does not perform unit testing, or even compile the code. The testing process is completely separate from the development process. All Cleanroom developed code is subject to mathematical verification techniques to verify software correctness. By reasoning about the effect of each code statement, individually and then in combination, the developer can prove the developed code meets specification [Linger]. The goal of Cleanroom is to impose discipline on software development so that system correctness results from a cohesive, readable design rather than reliance from execution based testing [Linger].

3. **Statistically Based, Independent Testing**: Software testing in Cleanroom is viewed as a sampling problem [Whittaker and Poore] where statistical usage testing [Poore, Mills and Mutchler] for certification replaces coverage testing for debugging [Mills, Dyer and Linger]. An independent testing team simulates the anticipated usage of the system to determine the software reliability [Poore, Mills and Mutchler]. The testing team create usage models [Whittaker and Poore] which define all possible scenarios of software use along with their probabilities of occurrence. A random sample is selected from this model and run against the current system increment. The responsibility of the test group is to certify the reliability of the increments and software product rather than assist the development group in getting the product to an acceptable level of quality [Linger; Dyer].

**Developing Client/Server Applications**

Developing Client/Server (C/S) applications requires a significant shift from the single system development model. It is important to consider that your development platform is no longer a single operating system and architecture, but an entire network. By breaking down an application into functional pieces, each piece can be distributed throughout the network in order to maximize your applications capability and performance.
When a stand alone or one-tier application is moved into a C/S environment, various components of an application are distributed to different hardware platforms (referred to as "servers"). Figure 1 identifies the different components of an application and how they are distributed in one-tier, two-tier and three-tier environments. The main components of an application are:

1. A **presentation** or user interface layer that handles display requirements. This layer includes the GUI and associated menus, displays, and the flow of screens and dialog boxes with which the user interacts.
2. The **application or business logic** layer handles all the application processing that needs to be performed. This may include calculations, data analysis or logical record handling.
3. The **data** layer handles all retrieval and storage of information. Usually, this layer will consist of the relational database and all processes needed to interact with the database.

In a standalone application, the user interface, business logic and database reside on the same machine. When an application is divided into two-tiers, the user interface and business logic reside on the client (referred to as "fat client") while the database layer resides on the "database server." In the three-tier environment, the user interface resides on the client, the business logic resides on the "application server" and the database resides on the "database server."

**Focus of the Research**

The focus of this research is to define specific design, development and testing requirements of three-tier C/S applications. After these issues are identified, we will determine how CSE can be used to help support C/S development, or determine any deficiencies in CSE and suggest enhancements. Specifically we will investigate:

- Application partitioning. How to model the components of the C/S application with respect to user interface, application process logic and application data. Since components of the application will reside on different servers within the network, network characteristics will have a major impact on the overall success of the application. These issues should be addressed at application design time.
- Data Modeling. In three-tier C/S applications, data will reside on a separate database server or multiple database servers. Data can also be represented as parallel data, replicated data or distributed data. We will investigate the effect of the different data models on C/S application design.
- Certification/Testing. Three-tier C/S applications require a different certification strategy than stand alone applications. In this research, we will investigate different certification/testing strategies in the C/S environment. Specifically, how the certification process in CSE can be applied to three-tier C/S applications.

**Potential Use of Research Results**

In addition to defining unique three-tier C/S development issues, this research is expected to provide new insights into developing applications in a multi-tier environment. By suggesting enhancements to CSE, this research will provide application developers with a development methodology to help create higher quality and more reliable C/S applications.

**References**


