Quality, Cleanroom and Formal Methods

Zarrin Langari
School of Computer Science
University of Waterloo
Waterloo, On, Canada
1-519-888-4567
zlangari@uwaterloo.ca

Anne Banks Pidduck
School of Computer Science
University of Waterloo
Waterloo, On, Canada
1-519-888-4567
apidduck@uwaterloo.ca

ABSTRACT

We have proposed a new approach to software quality combining cleanroom methodologies and formal methods. Cleanroom emphasizes defect prevention rather than defect removal. Formal methods use mathematical and logical formalizations to find defects early in the software development lifecycle. These two methods have been used separately to improve software quality since the 1980's. The combination of the two methods may provide further quality improvements through reduced software defects. This result, in turn, may reduce development costs, improve time to market, and increase overall product excellence.

Defects in computer software are costly. Their detection is usually postponed to the test phase, and their removal is also a very time consuming and expensive task. Cleanroom software engineering is a methodology which relies on preventing the defects, rather than removing them. It is based on incremental development and it emphasizes the development phase. An enhancement to this methodology is presented in this paper, which combines formal methods and cleanroom. The efficiency of the new model rests on an appropriate logical representation, to write the specification of the intended system. In the new model, design plans are formally verified before any implementation is done. The advantages of finding defects in the early stages are decreased cost and increased quality. Results show that, by using formal methods, a higher quality will be achieved and the software project can also benefit from the existing mechanized tools of these two techniques.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification – *formal methods.*

D.2.8 [**Software Engineering**]: Metrics – *process metrics, performance metrics*

General Terms

Measurement, Performance, Verification.

Keywords

Software Quality, Cleanroom, Formal Methods.

3-WoSQ '05, May 17, 2005, St Louis, Missouri, USA.

1. INTRODUCTION

Failures in a software project are costly, no matter whether these failures happen during development or later on the customer's site. There are many examples where systems failed to operate as the result of software or hardware failure. Peter G. Neumann in Software Engineering Notes [7] stated: "A computer fault may have accidentally erased some immunization records from among 425,000 Toronto school children during April 2002... This is especially important since failure to ensure appropriate immunizations can possibly result in suspension of children from school." He also pointed to the grounding of Air Canada "Jazz" airline by a computer virus in a flight-planning computer in early February 2003. Another example is Therac-25; a computer-controlled radiation therapy machine made by Atomic Energy of Canada which overdosed six people between June 1985 and January 1987. The error was a timing problem on data entry. The program did not consider data entry corrections made by the operator.

The recovery cost of these failures is huge. There are some direct costs related to quality problems [2]:

- many tests to find defects;
- repeating tests after each error correction;
- customer's requirements do not match the system functions after delivery;
- delays in marketing, idle resources, staff redeployments;
- postponed new developments, for maintenance;
- good developers assigned to do error correction;
- other products may be postponed if the direct customer is another department waiting for the product.

Besides the cost, government legislation mandates safe software and improved methods to gain defect-free developments in safety-critical systems. For example, the European Commission legislation, the Machine Safety Directive, effective from January 1993 states: if there is an error in the machine's logic that results in injury then a claim can be made under civil law against the supplier [1]. The manager can be charged for criminal acts if there is proved negligence in a product's design and manufacture.

This can result in sending the manager to jail for several months. Therefore software costs could be millions of dollars and also many human lives.

A possible solution is using *Cleanroom* techniques. These techniques have been used as development methods previously. Nevertheless, they did not address issues raised in the specification and requirement phase. This paper presents an enhancement to the previous Cleanroom methodology using *Formal Methods*. This enhancement considers applying formal techniques to the specification phase of Cleanroom methodology. The paper also outlines the quality and cost benefits of the new approach.

Section 2 introduces the Cleanroom methodology and how management uses this methodology. Section 3 describes the need for an enhancement to Cleanroom methodology. Later, the section introduces formal methods, their usage in the new enhancement, and the automation of correctness proofs by these methods. Sections 4 and 5 investigate the influence of formal methods on quality and cost. The paper ends with a summary of the points covered.

2. CLEANROOM METHODOLOGY

Semiconductor plants use clean rooms to produce failure free products. In clean rooms, people wear sterile gowns and masks to manufacture integrated circuits. In the process of producing circuits, a speck of dust can be considered a failure. Any failure is considered a failure in process rather than in product. Errors are tracked to process failures, failed products are thrown away, and the process is fixed [8].

Mills has developed the Software Cleanroom Methodology [6], which is an approach emphasizing defect prevention over defect removal. A failure in the software is considered a process failure, which can be in software specification, design, or verification. The process is fixed and the failed product is thrown away. In software this means sending the erroneous unit back to the point in process where the failure happened. Cleanroom development uses the Waterfall software model as its base, and adds incremental development to the traditional model [5]. The objective of the Cleanroom software engineering process is to develop near zero-defect software.

There are four teams operating in Cleanroom software process. These teams may turn into multiple teams of teams for large projects: *The Specification Team* analyzes and represents customer requirements. It produces two specifications: *functional (requirement)* and *usage*. The functional specification defines the required external system's behavior in all circumstances. External behavior is a function mapping of all inputs to outputs of the system. The team considers all input cases including unexpected and erroneous inputs, to express the system's external behavior. This formalization will then be used instead of the natural language description of what is required that can be easily misinterpreted and is often incomplete. Usage specification defines usage scenarios and their probabilities for all possible system usages. The functional specification is the basis for incremental software development, and the usage specification is the basis for generating test cases for incremental statistical testing and quality certifications [4].

The Development Team carries out incremental analysis and design activities to produce a formal design in box structures. Designs are verified to be correct through mental proofs of correctness in team reviews. The team builds a plan in which the software will be incrementally built and certified. Each increment is a working program, and a specification is produced for each increment. The first increment provides a basic functionality and each successive increment adds functionality until the development cycle specification is satisfied. Each increment will undergo the design and build, functional verification, and certification phases. The logic of increments rests on breaking down the complexity of the system. Fixes and updates can also be done more easily in increments.

The Certification Team develops test cases to test the functionality of the system stated in the increment specification. Even though the code has been formally verified, it is still necessary to test the quality of the software. This testing approach is different with traditional testing, in which testers assume there are errors in the software and try to find as many as possible. Certification, which is the Cleanroom term for testing, certifies the software reliability and performance. It does not seek errors in the software. This type of testing focuses on external system behavior not the internal part of the software. Cleanroom certification and statistical usage-based testing, does not measure quality in defects per line of code.

The Documentation Team produces documents in parallel with the development and certification teams. For each increment the evolving document will be validated for quality.

2.1. Cleanroom Management

Management planning and control in Cleanroom is based on developing and certifying software increments. The increments are developed and certified by small, independent teams. Determining the number and functional content of increments is an important task. For each increment, the development team carries out a design and verification cycle, based on the functional specification. Completed increments are periodically delivered to the certification team for statistical testing, and errors are returned to the development team for correction. The idea is to quickly develop the right product with high quality for the user, then go on to the next increment to incorporate new requirements arising from user experience.

In Cleanroom, box structures allow accurate definition of the required user functions without dealing with the internal behavior of boxes. Then the development team verifies the correctness of functionality against the requirements. Finally, verification reviews are held by the team to formally or informally verify the software using a set of correctness proofs. Proof of software correctness can be done by direct assertion of correctness conditions.

In traditional Cleanroom, there is no emphasis on using formal proofs or automated formal proofs to verify correctness of boxes. In addition, in traditional Cleanroom, the development cycle has the main role in the software life cycle. The next section

investigates the need for the enhancement of traditional Cleanroom considering the importance of the specification phase, and the use of formal methods.

3. ENHANCED CLEANROOM

Cleanroom development begins with a specification of required system functions. Without rigorous specification technology, it is difficult to devote time and effort to the specification process. Specifications are normally written in natural language, with inevitable ambiguities and omissions. In addition, in a box structure, it is important to define specifications correctly. Therefore there is a need to translate the natural language to a *formal specification*. Then all possible circumstances of usage, such as input/output paths, can be verified against the formal specification of the system. It takes months or even years to perform proofs for a small or medium sized industrial project [3], so formal software development is impracticable without appropriate automated techniques. In addition, programs may contain an infinite number of paths that cannot all be checked manually.

The purpose of a formal specification is to provide an unambiguous notation that can be validated. The efficiency of using *Formal Methods* relies on the choice of an appropriate logical representation, which eases the natural specification of the intended system and the proofs being done. Most specification languages used in automated formal techniques can specify, at a minimum, propositional logic.

In the next sections, we define Formal Methods and their influence on software quality and costs.

3.1. Formal Methods

Formal methods use a mathematical and logical formalization to prove that key properties of the system satisfy the expected behavior of the software system. Characteristics of formal methods include [9]:

- Formal methods check the consistency of the system's descriptions. They make sure properties that the system analysts have defined meet the requirements of the system. They actually check whether the analysts have correctly interpreted the system's requirements.

- Formal methods make it possible to find defects in the system early in the software lifecycle. Due to early defect detection the correct implementation through consistent requirements is possible.

- Formal methods avoid more testing. After applying these methods to high quality software systems, they find defects that may go undetected after extensive testing.

- Formal methods use mathematical notations to formalize the system's descriptions. By using mathematical notations (e.g. $\forall$ for all), we can make sure the system is correct for all possible inputs. Test cases always check the system just for a finite set of inputs, but formalization allows a large (potentially infinite) set of inputs to be considered for correctness proofs.

- Formal methods can be present in all phases of the software project. The software project manager decides when these methods should be used in the analysis, design and development phases, to detect more defects. Many times these methods guarantee defect-free software.

Based on the above description of formal methods and the goal of Cleanroom methodology, we propose that using rigorous formal specification in the initial phases of the Cleanroom process can ease correctness verification and automation of this process. A difference between the traditional Cleanroom and the new enhancement is that design choices are formally verified before any code is implemented in increments. This has the advantage of finding design problems early and, hence, lowering the cost.

In the Cleanroom process, formal methods should be applied in two stages. First, they are necessary in the specification phase for specifying the system behaviour with a logical notation which is a basis for increment specification. Second, they can be applied in an iteration where each increment's design will be formally modeled and verified against its formal specification using an automated model checker.

3.2. Automating Correctness Proofs

Formal specifications use mathematical language to specify what a system is supposed to do. They use abstraction to remove details as much as possible. After formalizing the system's properties, we need to prove that these property statements are valid. A proof is a set of rules to justify what we conclude from a set of assumptions. There are automating tools, which help to provide formal proofs. From the definition, it may be concluded that everything has to be proved correct. In fact, however, many current industrial uses of formal methods involve no, or minimal, proofs [1]. This involves using a theorem prover e.g. Z notation[1] at the first cycle of the software project to have a high-level specification of the system to be designed. Then at the development level applying formal methods can be done by VDM (Vienna Development Method), which uses a set of rules to refine the operations and data structures in the requirement specification to reach an implementation level. Actually formal methods tend to reduce human involvement in evaluating arguments. They limit the acceptability of arguments to calculation, which can be checked mechanically [4].

The goal of formalization with the cleanroom methodology is error reduction in the early phases, enhancing quality, lowering cost and time to market. To achieve this goal, formal methods should be applied correctly and through the right choice of management for the level of formalization and tools.

In the following section, we discuss the influence of using formal methods on quality and cost.

---

1 A first order logic and set theory with graphical representation. Its use resulted in two awards for technological achievement: for the IBM CICS project and for a specification of the IEEE standard for floating-point arithmetic.

4. IMPROVED SOFTWARE QUALITY

The goal of cleanroom methodology is achieving higher quality rates. A traditional project may experience five errors per thousand lines of code (KLOC) in function testing for example. Considering the first execution and unit testing, it may increase to 25 errors/KLOC. Table 1 [10] shows faults discovered during unit testing for the delivered code based on different formal methods design types. Using this table, Hatton reports that the faults discovered during unit testing occur more often in informally designed modules.

Table 1. Faults discovered during unit testing

	FSM	*VDM*	*VDM/CCS*	*Total formal*	*Informal*
Number of faults discovered	43	184	11	238	487
Number of modules with this deign type	77	352	83	512	692
Number of faults normalized by the number of modules	0.56	0.52	0.13	0.46	0.70

Table 2 [10] compares the failure rate of projects that used formal methods and those that did not use formal methods.

Table 2. Failure rates reported in literature

Source	*Language*	*Failure per KLOC*	*Formal methods used?*
Siemens operating system	Assembly	6-15	No
NAG scientific libraries	Fortran	3.00	No
CDIS air-traffic-control support	C	0.81	Yes
Lloyd's language parser	C	1.40	Yes
IBM Cleanroom development	**Various**	**3.40**	**Partly**
IBM normal development	Various	30.0	No
Satellite planning study	Fortran	6-16	No
Unisys communication software	Ada	2-9	No

Others have proposed joint use of formal methods and rapid prototyping [11]. In software systems with combined Cleanroom and formal methods, the correctness is increased through formal specification, design, and verification. All errors are accounted for from the first execution on, with no private debugging allowed. Errors left behind by the Cleanroom correctness verification tend to be simple mistakes easily found and fixed by statistical testing, not deep design errors.

Based on the above results, we conclude that formal design, combined with Cleanroom, can yield highly reliable code.

5. COST BENEFIT

The biggest payoff from the use of formal methods occurs in the early life cycle stages, given that errors become more expensive to correct as they proceed undetected through later development stages. Early detection leads to lower life cycle costs. In traditional software systems, the test phase and later maintenance are also very expensive and many expensive resources (developers) are needed to fix the bugs. Often tests must be repeated to check the correctness of programs. These products are not as reliable as when formal methods have been used.

One additional factor in reducing the cost is reusability. Like the software development itself, formal methods can benefit greatly from reusing assets. The abstract specifications and general theories can be reused on the other parts of the same project or in entirely different projects. This is especially true when mechanized forms of formal methods are employed.

On the other hand, there are some overhead costs regarding the training of staff to provide specifications, and to use formal methods tools. This cost will be covered later in the project by eliminating most of the test phase, and by using the knowledge of the staff for other projects.

6. CONCLUSIONS

Cleanroom in software is the methodology that prevents defects from happening rather than removing them after they've happened. The main focus of this methodology is on incremental development. It uses box structures to verify the correctness of properties for each increment against the specification for that increment.

A new enhancement to Cleanroom methodology has been proposed which focuses on the specification phase. Formal Methods are the techniques suggested to be used at the specification phase to write all the user's requirements in a logical and mathematical language. The benefits of having formal specifications are:

- Unambiguous language in comparison to natural language.
- Logic is able to define statements that consider all possible input values. This is significantly better than unit tests, which are usually able to test just a small subset of input data.
- Design choices can be formally verified before any implementation.

- Correctness verification can be done automatically through theorem provers and model checkers [4].

- Changes in software specifications can be handled more easily.

Quality improvement and cost reduction are two other benefits of using formal methods in Cleanroom methodology. The experiments in industrial projects have been used to show how the number of errors decreased when formal methods were used. A low failure rate results in higher reliability and quality. The cost will be reduced as well, because the unit testing has been eliminated and error correction in early phases is easier and cheaper than in later development stages.

Further research is needed to discover the role of other factors such as team size or CMM level. A formal implementation of our ideas could also provide more precise results.

7. REFERENCES

[1] Bowen, J. The Industrial Take-up of Formal Methods in Safety-Critical and Other Areas: A Perspective, In J.C.P. Woodcock and P.G. Larsen, editors, *Proceedings of FME'93: Industrial Strength Formal Methods*, LNCS 670. Springer-Verlag, 1993.

[2] Deck, M. An Introduction to Cleanroom Software Engineering for Managers, *Cleanroom Software Engineering Inc*., Boulder, CO, USA, 1995.

[3] Hutter, D., Schairer, A. Towards an Evolutionary Formal Software Development, *Proceedings of 16th Annual International Conference on Automated Software Engineering*, Nov. 2001, pp. 417 –420.

[4] Kemp, K. Formal Methods Specification and Verification Guidebook for Software and Computer Systems, Volume I: Planning and Technology Insertion, NASA, 1998.

[5] Linger, R.C., Hevner, A.R. Achieving software quality through Cleanroom software engineering, *Proceedings of the Twenty-Sixth Hawaii International Conference on System Sciences*, Volume: IV, 5-8 Jan. 1993, pp. 740 -748.

[6] Mills, H.D., Dyer, M. and Linger, R.C. Cleanroom Software Engineering, *IEEE Software*, September 1987.

[7] Neumann, P.G. Risks to the public in computers and related systems, *ACM SIGSOFT Software Engineering Notes*, Volume 28, Issue 3, (May 2003), pp. 5-9.

[8] Oshana, R. Quality Software via a Cleanroom Methodology. *Embedded Systems Programming Magazine*, Sept. 1996, pp. 36-52.

[9] Palshikar, G.K. Applying formal specifications to real-world software development, *IEEE Software*, Volume: 18, issue: 6, Nov.-Dec. 2001, pp. 89-974.

[10] Pfleeger, S.L., Hatton, L. Investigating the Influence of Formal Methods, *IEEE Computer***,** Volume: 30 Issue: 2, Feb. 1997, pp. 33 -43.

[11] Quemada, J. Formal Description Techniques and Software Engineering: Some Reflections after 2 Decades of Research, *Proceedings of FORTE 2004*, LNCS 3235. 2004, pp. 33-42.