ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

Improvisation of Fault Classification and Detection

Thilagavathi R, Anitha Raj B, Gajalakshmi N

PG Scholar, Dept. of Department of IT, K.L.N College of Information Technology, Pottalpalayam-630633, Sivagangai Dt., Tamil Nadu, India[1, 2]

Assistant Prof, Dept. of Department of Information Technology, K.L.N College of Information Technology, Pottalpalayam-630633, Sivagangai Dt., Tamil Nadu, India[3]

ABSTRACT: Software faults are a major threat for the dependability of software systems. When we intend to study the impact of software faults on software behavior, the issue of distinguishing faults categories and their frequency distribution arises immediately. For this clear detection, clear classification is needed. Very little is actually known about the types of faults that programmers insert into their software. It is becoming more important that these faults are classified into different categories. In this project, a programming technique is implemented where programmers are required to categorize their faults at each iterative build of the software build cycle. Experiments were carried out that measured the number of faults at each build both using this technique and not using this technique. The result suggests that requiring programmers to categorize their faults during the software build cycle decreases the total number of faults in a program. Then Faults are detected based on their classified types. To provide enhanced detection, an efficient graph mining technology is implemented and loc counts are individually features for localization of faults. The project suggests programmers to categorize their faults during the software build cycle and then detecting them will decrease the total number of faults in a program.

KEYWORDS: Include at least 5 keywords or phrases

I. INTRODUCTION

This document is a template. An electronic copy can be downloaded from the conference website. For questions on paper guidelines, please contact the conference publications committee as indicated on the conference website. Information about final paper submission is available from the conference website.

Software maintenance is extremely important activity in software development life cycle. It involves a lot of human efforts, cost and time. Maintaining software includes Fault detection, Fault correction and Fault prevention. The real impact of Fault on software has gained substantial attention due to sophisticated and complex applications, commercial hardware, clustered architecture and artificial intelligence. Software fault detection has been studied in context of Fault prone modules, self healing systems, developer information, maintenance models etc. A lot of things like modeling and weight age of impact of different kind of faults in the various types of software systems classification has become most important process in software life cycle.

The impact of the faults on software is heavy and hectic. So proper dealing with the faults while handling software will give fruitful results. Detecting the faults that occur on software may be in the code, design or in the specification. Normally, in general programmers relay on testers for error corrections and the testers perform the required types of testing. But when faults are inspected under a certain types of classification at earlier stages of software build cycle, detecting them get more and more easy and the count of faults reduces. Leaving the count of fault occurrences apart, just their classification should be noted in the earlier cycle and later the faults should be detected based on the classified part. Focusing on the detection part, the occurrence of the fault is concentrated on the flow of the corresponding modules. Thus, an efficient disassembler known as Gnoloo disassembler clearly disassembles the program flow into the graphical structure of the methods called and the methods being called. This shows the end user the exact flow of the various modules that happens in the entire system. This type of enriched detection of faults will help the users who get the output but not the one he/she actually desired for.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

If even the disassembled work went right without any improper flow, detection is then programmed to focus on the number of lines physically executed. This type of inspections relies with the conformance that all the statements, branching conditions, exceptions, comments are all gone through. The implications of algorithm, if posted any are further detected to check if the proper implementation of the algorithm is done. Experiments are carried out with prior classification of faults and without prior classification of faults. Results were very obvious showing that faults, when classified at earlier stages and detected based upon their classification, yield less fault counts.

This project implements the difference between the inspection and testing. The strengths of inspections are that a similar process can be applied to a wide range of documents. The method can be developed to find more defects earlier and be more effective by using the same method in the whole project or the entire company. When it comes to finding faults in the code it can be hard to choose between inspection and testing. The right one is depending on what they looking for. It can it be much easier to find faults in code standard and traceability with inspection than it is with testing

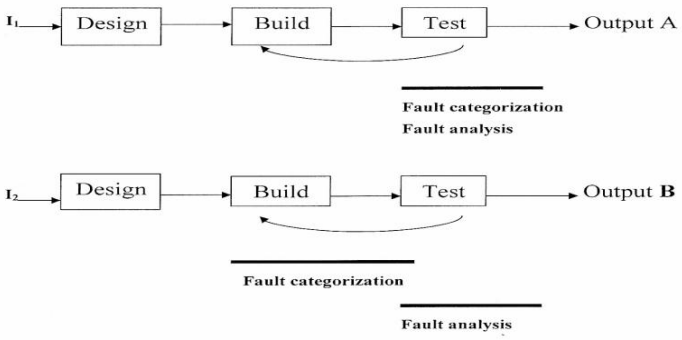

Figure1. Earlier Classification of Faults and Later Detection

II. REVIEW OF RELATED RESEARCHES

A handful of researches are available in the literature that deals with the software fault classification and detection. The existing system models for fault maintenance are simple and separate. There exists a separate model for fault classification and separate one for fault detection. The fault detection methods were in general based on modeling structures and estimation methods.

Existing Fault detection systems

- Process fault detection based on modeling and estimation methods.
  - Quantitative model based approaches.
- Artificial Neural Networks (Motor Incipient fault detection).
  - Fuzzy Logic.

Existing Fault Classification methods

Knuth's Method of classifying the Software faults.

- Beizer's bug taxonomy.
- Gray's classification of Software Fault.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

III. PROPOSED SYSTEM

We propose a system that makes programmers easy and effective to detect faults and isolate their relative frequencies. As said earlier, the systems proposed for fault detection and classification are unique and separate. We hence propose a combined system for both detection and classification. In detection, we have implemented a GNOLOO DISASSEMBLER to disassemble the regular sequential coding into a graph that indicates the list of the modules present and the graph of the modules called and the calling modules. The system also indicates the user to check in if all the statements are properly executed – including the branching (if, for, while, do etc), comments, looping, headers and all required and non-required lines.

The Algorithmic comparison is also made so that the proper implementation steps are checked and followed. Any faults that are injected in the software program are identified and the line in which the fault has been injected is exactly shown. If the injection made by the user is required for the implemented system, then the changes are accepted and option is available to fix and use the system with proposed changes. In the classification concept, we choose the most three essential conceptual classification of the software faults- the Code, Design and the Specification. In this way the experiments are carried out with and without classification and it resulted with – Classifying faults at earlier stage decreases the number of faults.

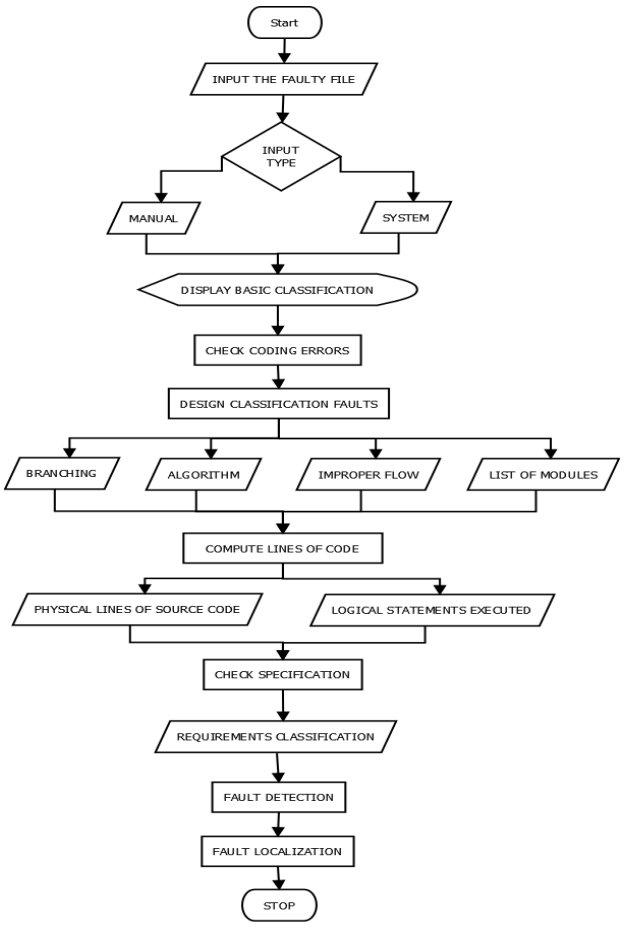

Figure2. Flow Diagram

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

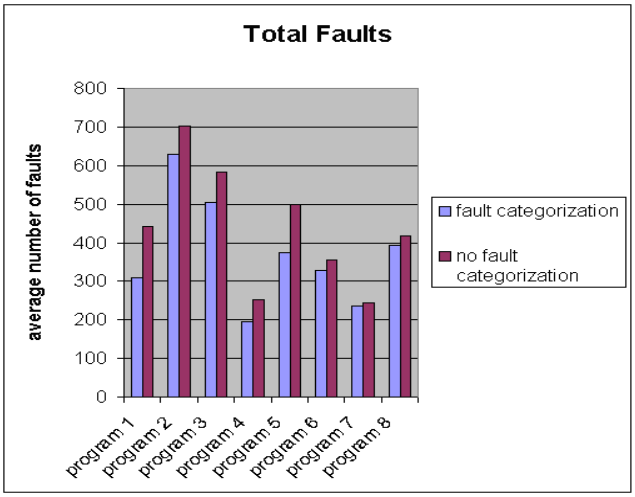

Figure3 Comparison With And Without Fault Categorization

A. GNOLOO DISASSEMBLER

In programming terminology, to disassemble is to convert a program in its executable (ready-to-run) form (sometimes called object code) into a representation in some form of assembler language so that it is readable by a human. A program used to accomplish this is called a disassembler, because it performs the inverse of the task that an assembler does. Disassembly is a type of reverse engineering. This Gnoloo disassembler compiles the original java file and Gets the class file as input. The disassembler disassembles the class file and creates the j file for the given input of a class file. This .j file contains the overall description of what is happening in the given program. Starting from the name of the declared class, each and every line in the coding are described in the disassembled .j file. This also clearly specifies the methods present in the given program and the methods that call the other methods or functions. From this information, a detailed graph is generated so that the list and the original flow of the process are correctly bought according to the coding. This disassembler was invented by Engel and a coexisting assembler called Oolong is also equally preferred in the industry.

B. Fault Inspector

- The Fault-Inspector can receive the fault injected program or the program under test from the user.
- The Fault-Inspector should ask the user what the condition of the current program is.
- The Fault-Inspector can provide the user with the classification page.
- The Fault-Inspector should keep the progress of the disassembled file and get the user know about the list of the modules and what flow the modules make with each other.
- The Fault-Inspector should check for the execution required for each and every line in the Program.
- The Fault-Inspector should keep in record with the algorithm and the steps to implement those algorithms.
- The Fault-Inspector should control the Gnoloo disassembler

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

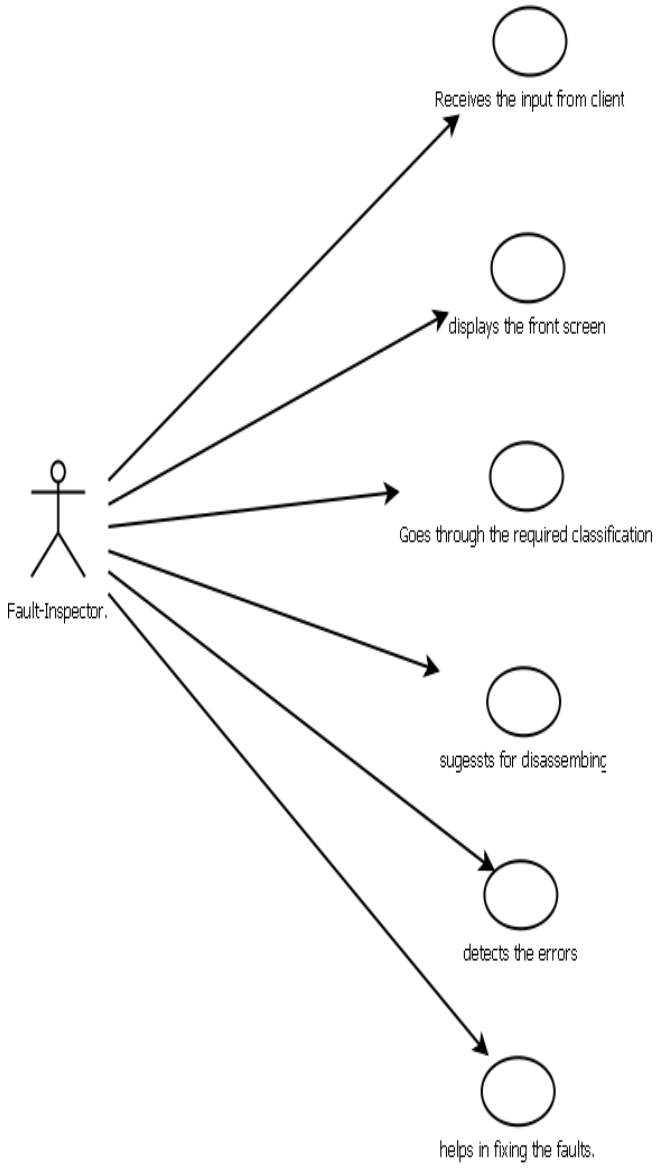

Figure4 Use Case – Fault Inspector Side

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

C. Client/User

- The client first looks at the Classification page.
- The client figures out the required classification and intends to check the coding.
- Later, the client should go in for the checking process of looking at the flow of the modules called.
- The clients can look for detecting the faults at the branch executions.
- The clients can also prescribe the algorithm he/she has implemented in the program and can ask the Fault- inspector to check if the prescribed algorithm has properly been implemented or not.

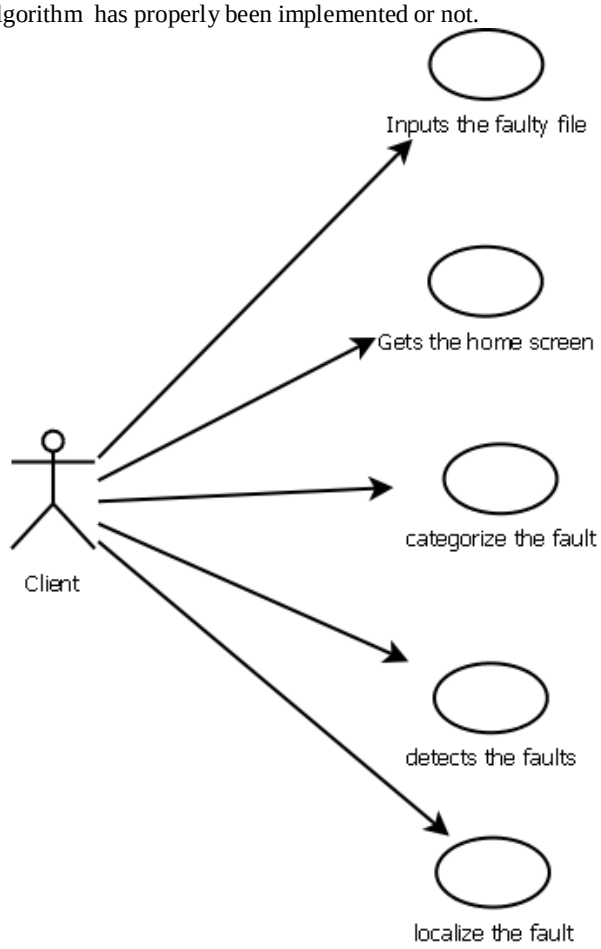

Figure5 Use Case- Client Side

IV. PROJECT DESCRIPTION

Implementation is the process of enlisting all the modules and executing it in the right order, using right software tools. The modules are the part of entire project executions that are implemented to provide the desired results. The modules involved in this project are listed as follows.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

A. Modules

- Classification of the Faults
  - Detect Code Faults
  - Detect Design Faults
  - Detect Specification Faults
- Code and Design Faults Categorization
- Disassembling the File
  - Use Gnoloo Disassembler.
  - Intake of Faulty File
  - Creation of .j File
- Branching Statements
  - ecution of Branched statements
- Physical Execution of Lines
  - ll Lines of Code
  - Count the Physically Executed Lines of code
- Algorithm Check
- Specification Inspection
- Localization of the Fault

B. Module Description

Faults Classification

The Faults are classified on the three major categories. Detect Code faults, Design Faults and Specification Faults. Each and Every category is analyzed for detailed Inspection. This module serves as the home screen in which the user can select his/ her basic type under which the fault can be inspected. Then the faults are inspected based on the type of classification they select.

Code and Design Faults

Code faults are the easiest type of faults to locate and fix. Code faults are quite often responsible for compilation errors and our compilers usually help by giving suggestions as to what is wrong with the code. Examples of suggestions include "Missing semicolon" or "Undeclared variable name".

Design Faults, unlike code faults, do not cause a compilation error. The compiler is no help so it makes this type of fault more difficult to locate. It is also possible for a program to run to completion without seeming to

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

generate a fault. The problem is that the output generated is not what is expected. In other words, the program is syntactically correct but the outcome of the program is not what the programmer intended. Software engineers now have to be able to trace through a program at the semantic level in order to locate the fault. This type of fault is more difficult to correct than a code fault.

Disassembling the File

The Input file is first disassembled to receive the .j file. This file clearly lists out the execution of all the lines executed. It disassembles the methods and specifies the called methods and the calling methods. This in turn produces the flow of the entire program. This ensures the client with the actual process happening in the program. Now this flow is displayed to the client to detect the required module that has been changed. This disassembling is done by the Gnoloo Disassembler.

Branching Statements

The executions are ordered in sequence for the checking of all the branched statements being executed. This involves all the branching statements like while, do, for, if, etc. This is to ensure only the true part of the program coding is executed. In some exceptional conditions, the unnecessary parts may get executed and this point of view is inspected in this field. This will ensure the client that only the physically executed number of lines of code is legally in correct flow. The output of this program is the overall number of Lines of code and the physically executed number of lines of code, to ensure the true and false part execution.

Physical Execution of Lines

The Overall Lines of code is counted, to check if all the lines in the program are properly executed. This includes comments, branching, headers, braces, general syntax, etc. Two options are provided were the selected file and the edited are selected. The original file and the edited file are then compared so as to figure it out where the line of change has occurred and the physically executed. This also predicts the place where the faults are modified. There exists an option where the modified changes are essential for the particular program. In such situations, this option will consider the changes made as a valid one and would save and continue from the changed format.

Algorithm Check

The algorithm implemented by the client is analyzed and a general syntax of the prescribed algorithm is generated and both the original and implemented algorithm are compared with each other and if there is any unnecessary declarations , this process will find the improper specification of the implemented algorithm.

Specification Inspection

Specification faults are the hardest type of faults both to locate and to fix. The software engineer believes the software is running correctly. When the customer looks at the output from the software, it is discovered that there has been a misunderstanding between what the customer wants the software to do and what it actually does. Fixing these types of faults require design analyses and clarification of specific requirements. The first thing the software engineer must do is to go back to the requirements document and make sure it was interpreted correctly. If it was, then both the customer and software engineer must review exactly what it is the customer wants and the software process must begin again.

Localization of the Fault After all these processes have been executed, the
Faults are located in the aspect they are identified. Then the faults are then detected and figured out. These experiments are carried out with and without classification. Obviously, it gave up with the fact that the total number of faults decreased when properly classified in the earlier build cycle and then detected in the later cycle.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

V. FUTURE ENHANCEMENTS

Database Enrichment

When this specific method is applied with external database characteristics, this would serve as an efficient model for the fault-less coding implementation.

Network Security

When the implemented project is accessed across a complex network, this will give rise to efficient fault tolerable Heavy networking connectivity.

VI. CONCLUSION

The prescribed methodology makes the programmer stop and think before trying to fix a fault. This approach of fixing faults tends them to thoroughly think through the problem and come up with a reasonable design to fix the fault before they start to code. This prevents overloaded errors and complex flow tracking. When the software engineer takes the time to think the problem thorough and understand why the software is functioning the way it is, both a quicker and better solution to the problem is usually found. This will be serving as the fastest and efficient approach to deal with the various faults to fix. This procedure is less time consuming. The detection process, which is highly descriptive and complex, is also self- explanatory. Results from this experiments show that there were fewer faults in a programming set of codes where the fault categorization is done than in a program where the fault categorization is not done.

REFERENCES

[1] S. M. Metev and V. P. Veiko, Laser Assisted Microtechnology, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.

[2] J. Breckling, Ed., The Analysis of Directional Time Series: Applications to Wind Speed and Direction, ser. Lecture Notes in Statistics. Berlin, Germany: Springer, 1989, vol. 61.

[3] S. Zhang, C. Zhu, J. K. O. Sin, and P. K. T. Mok, "A novel ultrathin elevated channel low-temperature poly-Si TFT," IEEE Electron Device Lett., vol. 20, pp. 569–571, Nov. 1999.

[4] M. Wegmuller, J. P. von der Weid, P. Oberson, and N. Gisin, "High resolution fiber distributed measurements with coherent OFDR," in Proc. ECOC'00, 2000, paper 11.3.4, p. 109.

[5] R. E. Sorace, V. S. Reinhardt, and S. A. Vaughn, "High-speed digital- to-RF converter," U.S. Patent 5 668 842, Sept. 16, 1997.

[6] (2002) The IEEE website. [Online]. Available: http://www.ieee.org/

[7] M. Shell. (2002) IEEEtran homepage on CTAN. [Online]. Available: http://www.ctan.org/tex-archive/macros/latex/contrib/supported/IEEEtran/

[8] FLEXChip Signal Processor (MC68175/D), Motorola, 1996