NASA-CR-200246

NCCW-0040

INTERIM

IN-61-CR

OCIT

8868

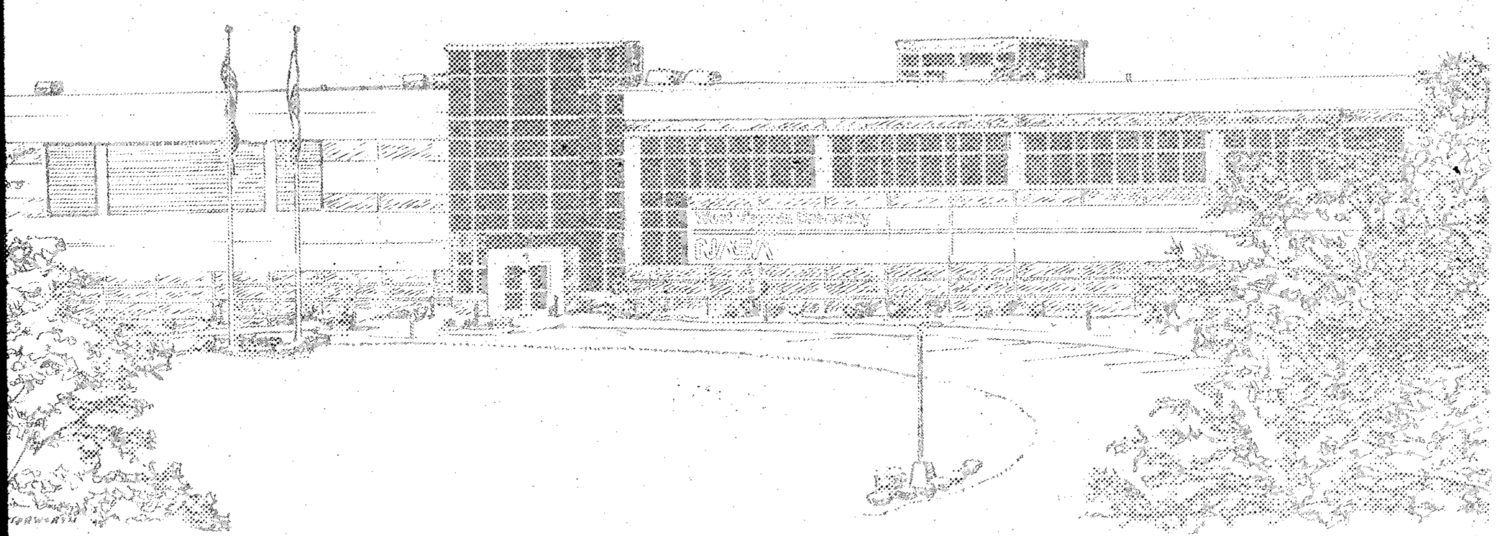
P. 41

NASA/WVU Software IV & V Facility
Software Research Laboratory
Technical Report Series

Annual Report

NASA/WVU Software Research Laboratory

1995



National Aeronautics and Space Administration

West Virginia University

According to the terms of Cooperative Agreement #NCCW-0040, the following approval is granted for distribution of this technical report outside the NASA/WVU Software Research Laboratory

George J. Sabolish

George J. Sabolish Date 1-30-96
Manager, Software Engineering

[illegible] 1/30/96

John R. Callahan Date
WVU Principal Investigator

Annual Report

NASA/WVU Software Research Laboratory

1995

Contents

1 Introduction **3**
- 1.1 Overview . . . 3
- 1.2 Background . . . 4
- 1.3 Objectives . . . 5
- 1.4 Approach . . . 6
  - 1.4.1 Software Processes . . . 6
  - 1.4.2 Metrics . . . 7
  - 1.4.3 Formal Methods . . . 7
  - 1.4.4 Software Tools . . . 8
- 1.5 Acknowledgments . . . 9

2 People **10**

3 Support Activities **12**
- 3.1 NASA/WVU Technical Lecture Series . . . 12
- 3.2 V&V Research Quarterly . . . 13
- 3.3 Technical Report Series . . . 13
- 3.4 World-Wide-Web Pages . . . 13

4 Process **14**
- 4.1 IV&V within Rapid Software Development . . . 14
- 4.2 IV&V Within Reuse-Based Software Engineering . . . 15
- 4.3 V&V Tools Under Development . . . 16
- 4.4 The Emerging Paradigm . . . 17
- 4.5 IV&V Effectiveness . . . 18

5 Metrics **21**
- 5.1 Defining Metrics for the ROSE Project . . . 21
- 5.2 Misrepresentation of Software Project Statistics . . . 22
- 5.3 Managing Software Projects Using the WWW . . . 25
  - 5.3.1 An Overview of WISE . . . 26
  - 5.3.2 A Brief Description of WISE . . . 26

5.3.3 Current Status of WISE 27

6 Testing **29**

6.1 Validation Testing – An Automated Approach 29

6.1.1 Definitions 30

6.1.2 Recommendation 30

6.1.3 Implementation 31

6.2 Requirements-based Testing and Analysis 31

6.3 The SRL Testing Toolkit 33

7 Classification **35**

7.1 Verification and Validation of RMP 35

7.2 Status Update for RMP 37

Chapter 1

Introduction

In our second year, the NASA/WVU Software Research Lab has made significant strides toward analysis and solution of major software problems related to V&V activities. We have established working relationships with many ongoing efforts within NASA and continue to provide valuable input into policy and decision-making processes. Through our publications, technical reports, lecture series, newsletters, and resources on the World-Wide-Web, we provide information to many NASA and external parties daily.

This report is a summary and overview of some of our activities for the past year. This report is divided into 6 chapters: Introduction, People, Support Activities, Process, Metrics, and Testing. The Introduction chapter (this chapter) gives an overview of our project beginnings and targets. The People chapter focuses on new people who have joined the Lab this year. The Support chapter briefly lists activities like our WWW pages, Technical Report Series, Technical Lecture Series, and Research Quarterly newsletter. Finally, the remaining four chapters discuss the major research areas that we have made significant progress towards producing meaningful task reports. These chapters can be regarded as portions of drafts of our task reports.

1.1 Overview

This Annual Report outlines specific research work on projects in four major areas of IV&V: *Process, Metrics, Testing, and Classification.* The following table shows a breakdown of these topic areas, subtopics, and outreach research projects that are exploring aspects of each subtopic. Many of the related project overlap into several topis and subtopics, but this is to be expected since it is difficult to separate the complex dimensions of software projects. Applied research into each of these areas is needed to support findings and recommendations that will form the Task Reports in each of these areas. Our pro-active approach has helped advocate the benefits of IV&V and demonstrate its effectiveness to NASA through our outreach efforts.

Topic Area	Subtopic	Related Project(s)	Page
Process	IV&V and Rapid Software Development Methods	Rapid Development Lab at Johnson Space Center (JSC)	14
	IV&V and Software Reuse	Reusable Objects Software Environment Project at Johnson Space Center (JSC)	15
	IV&V Tools	EOSDIS IV&V effort by Intermetrics, Inc.	16
	IV&V Process Paradigms	summary of other process efforts	17
	IV&V Cost Effectiveness	EOSDIS, International Space Station (ISS)	18
Metrics	Defining Metrics for the ROSE Project	ROSE at JSC	21
	Misrepresentation of Software Project Statistics	summary of other metrics efforts	22
	Managing Software Projects on the WWW	ROSE, EOSDIS, ISS	25
Testing	Validation Testing	EOSDIS	29
	Requirements-based Testing and Analysis	ISS	31
	The SRL test toolkit	EOSDIS, ISS, and the Cassini project at the Jet propulsion Lab (JPL)	33
Classification	V&V of the Reliable Multicast Protocol	EOSDIS, ISS, Information Sharing Protocol (ISP) at Johnson Space Center (JSC), and Cassini	35

Sections of each chapter in this report discusses the specifics of each project, topic, and the details of the outreach efforts involved. Further information can also be found va our Home Page on the World-Wide-Web (WWW) at **http://research.ivv.nasa.gov/**.

1.2 Background

Space is the most difficult environment for humans and machines to operate in. The vacuum of space is subject to enormous temperature changes, and our ability to fix spacecraft in orbit is extremely limited. With each new mission, NASA confronts increasing levels of system sophistication and complexity. In order to control and operate these new systems, a greater dependency is being placed on software. To meet the increased system demands, system software, in turn, is experiencing a dramatic increase in size and overall complexity. NASA recognizes that defects embedded within increasingly sophisticated and prolific system software represent a threat to future missions and has decided to take steps to minimize this risk.

In October 1991, the United States Congress passed Public Law 102-139 establishing a NASA Software IV&V Facility in Fairmont, West Virginia (hereafter referred to as "the Facility"). The primary mission of the Facility is to advance the state-of-the-art in the area of software Verification and Validation (V&V) tools, techniques and methodologies, and to advance and promote the Software Assurance discipline in general. To this end, the Facility is involved with Technology Transfer through the development of a comprehensive training program that will include involvement with community schools, technical training for software practitioners, management training for software and non-software program and project managers, and advanced training leading to graduate and postgraduate degrees.

The primary mission of the Software IV&V Facility's Research arm is to address the fundamental problems associated with preventing or reducing the number of defects in software products. Since "100% error free" software remains an unreasonable goal at this time, it is our mission to reduce the level of risk involved to some quantifiable level. In order to accomplish this, the research organization will identify, investigate and develop new software-oriented tools, techniques, and methodologies useful to both the Government and the commercial sector. The basis for identifying these new pursuits will stem from close ties formed between the NASA Facility, the academic world, and private industry. These strategic alliances will help to insure that new technological pursuits are consistent with the Facility goals and objectives and represent a valid investment in taxpayer dollars.

1.3 Objectives

The primary objectives of the research effort are to: (1) advance the state-of-the-art in the area of software Verification and Validation (V&V), including research, policy development, tools, techniques and methodologies; (2) promote the transfer of technology by incorporating a user based approach to the development of tools and prototypes that support research results and by working closely with the Technology Transfer Office in order to support their development of a world class training program; (3) ensure the industry relevance of research topics through consistent and active participation in selected internationally and nationally recognized software forums; (4) promote a true spirit of national competitiveness by soliciting candidate research topics through NASA Research Announcements (NRAs); (5) strive to eliminate duplication of effort through our strategic alliance with industry, academia, other government agencies and other NASA Centers by acting as a focal point for similar pursuits; and (6) to become a nationally and internationally recognized Center of Excellence for software research within NASA.

1.4 Approach

Although there exists a significant body of research in verification and validation of computer software, many problems still remain unresolved. Much of the existing research, however, indicates the need for improved specification and analysis of software requirements and design. For example, it is recognized widely that more emphasis must be placed on activities in the early phases of the software development life-cycle (i.e., requirements and design) because over 75% of software faults are committed in these phases. Yet, relatively little research has been done to solve this problem because traditional early life-cycle phases have been informally defined and practiced. This is an example of an area where future research in verification and validation can have an important impact.

Since development and V&V are complementary efforts, changes in one process inevitably affect the other. We expect that advances in V&V research will significantly influence development techniques and vice-versa. For example, development of a specification technique that proves effective in verifying timing constraints in real-time software will cause it to be used in future development efforts. Therefore, V&V research should not be limited to techniques applicable to current development methodologies, but rather they should significantly advance the state-of-the-art of V&V pro-actively relative to software development. The following are considered to be prime candidate focal areas of concentration to be factored into the formation of an overall general strategy for research to be performed at the Facility by West Virginia University as well as others to come.

1.4.1 Software Processes

IV&V plays an important role in software development by providing mechanisms for: (1) independent customer feedback, and (2) feedback into the development process. Although IV&V may add 15 to 45% to the cost of a software project, it may recoup these costs by delivering high-quality and avoiding costly problems after delivery and during the later parts of a product's lifetime (i.e., maintenance).

Tools alone do not comprise an IV&V team because they must fit into a process. First, process management tools should be developed to (1) track existing process practices and (2) automate process execution. Research in software process automation has been successfully applied to a few, select domains of well-understood problems, but much work is needed to apply automation effectively in many domains. Research activities will emphasize the establishment of robust management processes in order to provide for repeatability with respect to the application and evaluation or research products.

1.4.2 Metrics

It is necessary to assess the current needs and problems of software developers who are using or wish to use IV&V in their efforts. Many surveys exist regarding particular development methods, tools, and practices. The Facility and its related research efforts will strive to consolidate some of these studies into a repository and perform additional studies to determine the effectiveness of existing techniques. This repository will be a valuable asset for NASA not only in determining which techniques are effective but also in comparing existing efforts to past performances. In this manner, IV&V efforts can determine whether or not current projects are "on track" relative to similar projects. This means that new evaluation methods must be devised for assessing existing and new software development and IV&V methodologies. These evaluation techniques will be closely linked with the research studies of software process models (see below).

1.4.3 Formal Methods

One of the most important directions toward solving problems at early phases of the software development life-cycle involves the use of formal methods. A formal method is any technique for specifying and verifying precisely that software implementations agree with their specifications. The precision of a formal technique allows no room for misinterpretation and permits automated and repeatable analysis of the software specifications. Informal methods, on the other hand, leave room for different interpretations of the customer's intent.

As promising as formal methods seem, they have serious limitations. Indeed, one of the major barriers to the adoption of formal methods has been their high-cost of application due to the high-skills needed to write and analyze formal specifications. In addition, there is no guarantee that a formal specification is correct since they are written by human developers.

Formal methods research covers a broad spectrum of software development and V&V topics. Such research significantly impacts V&V research since increased precision would eliminate many of the common problems caused by current informal methods used in early life-cycle phases. Furthermore, different levels of rigor can be applied to projects depending on the cost-benefit of applying a particular technique. For example, projects involving possible loss-of-life or high-cost equipment (e.g., a satellite) might benefit from high-cost formal specifications. Much research is needed to leverage the advantage of formal methods into development and V&V while reducing the costs of its application. The Facility's research approach will strive to capitalize on the advantages that a formal methods approach offers, while at the same time trying to reduce the cost and complexity associated with its implementation.

1.4.4 Software Tools

A major part of the V&V research effort will be in the research and development of new software tools. These tools will incorporate or improve upon previous software tools at all phases of the software development life-cycle. Although software testing is still an important part of many validation activities, emphasis will be placed on those research areas that have been identified as having a high return-on-investment related to V&V. Particular emphasis will be placed on early life-cycle tools (requirements & design), software process modeling tools, tools to aid in the generation of integrated specifications sets, tools for configuration/interface management and software maintenance. The following paragraphs provide some important considerations that need to be addressed when developing software tools.

Because the use of a software tool often implies a particular class of software process models and representation schemes, our concept of software tools must evolve beyond the stand-alone, proprietary approaches of current computer-aided software engineering (CASE) tools. Our tools must work in a framework that accommodates existing commercial off-the-shelf (COTS) tools so that they can evolve with industry standards and work external to NASA and the IV&V Facility Research. These tools must be designed within various constraints so that they are compatible with other tools and can seamlessly evolve in the face of external pressures.

First, the software tools must address the needs of teams. Software development efforts are team-oriented efforts that involve the coordination of large groups of people. The application of IV&V to a software development effort implies that separate groups (development and V&V) must coordinate their efforts to make progress. In some cases, automated tools are a necessity to streamline procedures that would otherwise be prohibitively expensive to coordinate and perform in parallel.

Second, they must address the need for legacy. It is highly unlikely that developers will abandon current techniques immediately and adopt radically different policies. Therefore, it will be necessary to develop strategies for incremental adoption of new tools and techniques that accommodate legacy practices.

Third, they must address the need for maintenance. It is estimated that 40-60% of software project costs are in the maintenance phase alone. For long-term efforts like Space Station and EOS this will particularly be the case. In the maintenance phase, 60- 80% of the costs are for functional enhancements of the software. Existing tools are primarily geared toward a single execution of development phases (i.e., requirements, design, code, test). New tools and related techniques are needed to address long-term sustaining software efforts that will significantly reduce software costs in the long-run.

Finally, software tools must address the need for use in different processes. Different application domains (e.g., embedded avionics software, database systems, etc.) will require different process models that depend on the level of criticality of software, the level of assurance, the level of safety analysis, and the resource constraints of the project itself.

Our research approach will require research activities that develop tools or prototypes that focus on the above considerations and ensure that the quality of their products are commensurate with and complimentary to their research objectives and are consistent with their end-use functions.

1.5 Acknowledgments

I would like to thank the many people who have made the NASA/WVU Software Research Lab a success. Mr. George Sabolish, Manager of Software Engineering Research and our project contracting officer, has provided valuable input and assistance. He is truly an integral part of our research effort that makes the partnership between NASA and the University work on a daily basis. Ms. Kathyrn Kemp, Deputy Director of the NASA Fairmont Facility, has also provided invaluable interaction with the research team. Finally, Mr. Charles Mertz, Director of the NASA Fairmont Facility, has provided sound advice, guidance, and judgment in helping the research project set its goals and deliver valuable products. All of our NASA counterparts have played important roles as we seek to create research products based on actual, on-going software efforts.

On the research team itself, I would like to thank Todd Montgomery who has been with the project since the beginning. I hope he continues to make significant and important progress towards his Ph.D. degree. Many others have played roles shortly after the project began: Ramana Reddy, Steve Easterbrook, Edward Addy, Butch Neal, Frank Schneider, Wu Wen, Sudhaka Ramakrishnan, Yunqing Wu, Jeff Morrison, Wei Sun, Sudhir Koka, Nicholay Gredetsky, Aruna Sonti, and many other students.

In addition, I want to thank people external to the NASA/WVU SRL who have offered sound advice and guidance. These include: Nancy Leveson, John Gannon, John Knight, Victor Basili, Frank McGarry, and many others.

Finally, I want to thank Ms. Vivian Jenab, the administrative secretary for the NASA/WVU Software Research Lab, for her tireless efforts.

Jack Callahan
December 1995

Chapter 2

People

Several new personnel joined during our second year. These include:

Dr. Steve Easterbrook is currently a professor at the University of Sussex. He will join our group during a sabbatical for the next two years in Fairmont. Dr. Easterbrook is the author of several articles in the area of requirements engineering and has worked most recently with Dr. Anthony Finkelstein on a paper published in the 1995 International Conference on Requirements Engineering.

Mr. Edward Addy works in the areas of independent software nuclear safety analysis and software reuse. Working at Logicon, Inc., he performed nuclear safety analysis on Tomahawk Cruise Missile systems. He was task leader for developing the US Navy's Software Reuse Implementation Plan and conducted a pilot project in domain analysis for the Program Executive Office of Cruise Missiles and Unmanned Aerial Vehicles. He has served as a site manager for Logicon Technical Services. Prior to joining Logicon, Mr. Addy taught Mathematics and Physics at the Navy Nuclear Power School and the College of Wooster (Ohio). He received his B.S. in math education from Michigan State and his M.S. in mathematics from Wake Forest University.

Dr. Ralph Neal has worked in the software industry for over 20 years and will soon complete his Ph.D. at Virginia Commonwealth University. His research focus is on the measurement of software and the software production process. Dr. Neal worked in the Computing Services Department of West Virginia University for 15 years as an Administrative Systems Specialist. Dr. Neal is a graduate of West Virginia University where he obtained a bachelors degree in Computer Science. He has published papers on total quality management, measurement theory for software metrics and object-oriented software metrics.

Dr. Frank Schneider has over 20 years of experience in various aspects of software science in academia as a professor, in government, and in the industrial arena as well. Dr. Schneider has worked as a quality assurance engineer at NASA's Jet Propulsion Laboratory since 1988. During that time he has worked on standards generation and review, the Formal Inspection process and most recently as an Independent Test Engineer. In a discipline oriented approach he represented Quality Assurance as 1 of 9 disciplines in the design of the laboratory wide Systems Development Management Guide (D-5000). He has been involved in technology transfer of the Formal Inspection process at JPL and as a practitioner in a wide variety of verification and validation areas. As an Independent Test Engineer, he has authored a test plan for a subsystem for NASA's new 34m Bean Waveguide Antennas to be installed as part of the Deep Space Network at Goldstone, CA; Canberra, Australia; and Madrid, Spain. Dr. Schneider will locate to Fairmont as a senior member of the research team, and play a role in the Verification and Validation of software components of the Cassini Spacecraft.

Dr. Wu Wen has a background in robotics and is interested in mobile computing and reinforcement learning. He will be working with the NASA New Millennium Project.

Chapter 3

Support Activities

This chapter describes support work at the NASA/WVU Software Research Lab. These activities include our Technical Lecture Series, Research Quarterly (a newsletter), Technical Report Series, and our World-Wide-Web resources.

3.1 NASA/WVU Technical Lecture Series

Information on the Technical Lecture Series and the speakers can be found on the NASA/WVU SRL home page at http://research.ivv.nasa.gov.

June 19, 1995 - Dr. Mats Heimdahl (at NASA/Fairmont) of Michigan State University will talk about his work on TCAS II requirements that he performed while a Ph.D. student under Dr. Nancy Leveson.

July 7, 1995 - Dr. Frank Stomp (at NASA/Fairmont) of AT&T Bell Laboratories will speak about his work on formal verification and validation of complex telecommunication protocols.

July 19, 1995 - Dr. Frank Schneider (at NASA/Fairmont) of the NASA Jet Propulsion Lab (JPL) will speak about his work on Software Quality Assurance on Deep-Space Network ground support software projects.

July 20, 1995 - Dr. Eugene Zima (at WVU/Fairmont) of Moscow State University will speak at WVU on is work in the area of symbolic algebra and analysis.

August 11, 1995 - Dr. Walcelio Melo (at NASA/Fairmont) of the University of Maryland will speak about his work with Dr. Victor Basili in the area of software metrics.

August 14, 1995 - Dr. Sreeranga Rajan (at NASA/Fairmont) from the University of British Columbia will talk on his work at the Stanford Research Institute under Dr. John Rushby on formal specification and verification of software requirements.

August 16, 1995 - Dr. Roger Fujii (at NASA/Fairmont) of Logicon, Inc., a world recognized expert on IV&V, will talk about his experiences on major IV&V efforts and his visions for future IV&V research.

September 20, 1995 - Dr. John Yen (at NASA/Fairmont) of Texas A&M University spoke on his research to develop a methodology for providing intelligent assistance to requirements engineering.

December 15, 1995 - Amer Al-Rawas (PhD candidate at the University of Sussex, England) spoke on his approach to improving communication by identifying the weaknesses of different specification notations and compensating for these through the provision of annotations.

January 18, 1995 - Dr. John Knight (Department of Computer Science, University of Virginia) will present results from two case studies in developing software for safety-critical systems.

February 26, 1995 - Dr. Mark Klein (Applied Research Lab, Pennsylvania State University) will speak on addressing the weaknesses with current coordination technology.

3.2 V&V Research Quarterly

The NASA/WVU Software Research Lab publishes a quarterly newsletter, called the V&V Research Quarterly (VRQ), that is available at 304-367-8248 (Ms. Vivian Jenab) or most are available via our WWW home page (see below).

3.3 Technical Report Series

We have published over 30 technical reports in the Lab. These can be ordered at 304-367-8248 (Ms. Vivian Jenab) or most are available via our WWW home page (see below).

3.4 World-Wide-Web Pages

We provide extensive information on our activities on the World-Wide-Web (WWW). You can access information on the people, reports, and other publications listed in this chapter via our home page at `http://research.ivv.nasa.gov/`.

Chapter 4

Process

This chapter concerns work regarding software development and IV&V processes. The areas covered include research regarding IV&V in rapid development environments, reuse-based environments, tools, and research on overall process models between development and IV&V.

4.1 IV&V within Rapid Software Development

IV&V as its practiced today is almost exclusively in the domain of the waterfall life cycle model. Although the best IV&V is integrated into the development life cycle, the deliverables are almost always due at the end of one of the waterfall life cycle stages. At the Guidance, Navigation, and Control Rapid Development Lab (GN&C RDL), the Aerospace and Flight Mechanics Division of Johnson Space Center (JSC) is studying the rapid development process. How will the rapid development process affect the practice of IV&V?

The researchers of the NASA/WVU Software Research Lab traveled to JSC to witness the presentation of a proposed Research Technical Operations Plan (RTOP) designed to provide process definition for the rapid development of software. Several aspects of the proposed project are of interest to the researchers.

The first item of interest is the integration of Requirements IV&V into the rapid development lifecycle. Rapid development (as defined in the presentation) is characterized by using a "build-a-little, test-a-little" philosophy to provide early integration of all hardware and software elements which are then incrementally improved. Is it appropriate to perform IV&V on each release (often called a drop) of the iterative process? If we are to do IV&V on each drop, are we not doing extra work by pointing out the requirements that were left out on purpose because they were to be picked up in later drops? If we don't perform IV&V on each drop, is there another way to keep the independent verifiers in the development loop? Perhaps the answer will be to develop the requirements matrix and assign the requirements to "drops". No one knows, of course, so we'll have to keep you posted as the RTOP progresses.

The second item of interest is the verification of auto-coded source code. The fourth generation code generator currently being used in this project has the property of generating entirely reordered code when a seemingly minor change has been made to the data flow diagram from which the code is generated. Should the source code be verified each time it is regenerated? In an analogous situation, machine code is not verified each time the source code is recompiled; indeed the machine code is never verified. Compilers are certified and thereafter accepted as accurate. Perhaps the code generator should be certified. If an uncertified code generator is being used, what are the proper IV&V activities to insure the accuracy and completeness of the project?

The third item involves identifying measures that capture the dimensions of the process and the product. As a product measurement, lines-of-code has long been recognized as a misleading metric. With the acceptance of code generators (where the code may vary greatly from one drop to the next), lines-of-code becomes absolutely meaningless. Metrics designed to measure structured systems have had limited success in measuring object oriented systems. The rapid development paradigm adds another dimension to the problem. How do you measure a project that is by definition only partially complete? The RTOP plans to emphasize function points as desirable units in which to measure productivity. Function points may or may not turnout to be the final solution but they are almost certainly a good first approximation of the units to measure.

4.2 IV&V Within Reuse-Based Software Engineering

Verification and Validation (V&V) methods have been used to increase the level of assurance of critical software, particularly safety-critical and mission-critical software. This activity is often performed by an agent independent from the developing organization, and is then called Independent V&V. Software V&V is a systems engineering discipline that evaluates the software in a systems context.

The V&V methodology has been used in concert with various software development paradigms, including waterfall, spiral, and evolutionary development, but always in the context of developing a specific application system. However, the reuse-based software development process separates domain engineering from application engineering in order to develop generic reusable software components that are appropriate for use in multiple applications.

The earlier a problem is discovered in the development process, the less costly it is to correct the problem. To take advantage of this, V&V begins verification within system application development at the concept or high-level requirements phase. However, a reuse-based software development process has tasks that are performed earlier, and possibly much earlier, than high-level requirements for a particular application system.

In order to bring the effectiveness of V&V to bear within a reuse-based software development process, V&V must be incorporated within the domain engineering process. To date, no model has been developed for this. Failure to incorporate V&V within domain engineering will result in higher development and maintenance costs due to losing the opportunity to discover problems in early stages of development and having to correct problems in multiple systems already in operation. Also, the same V&V activities will have to be performed for each application system having mission or safety-critical functions.

On the other hand, it is not possible for all V&V activities to be transferred into domain engineering, since verification extends to installation and operation phases of development and validation is primarily performed using a developed system. This leads to the question of which existing (and/or new) V&V activities would be more effectively performed in domain engineering rather than in (or in addition to) application engineering.

The Reusable Objects Software Environment (ROSE) is a new software development paradigm adopted by the NASA JSC Mission Operations Directorate. This process is intended to reduce software maintenance costs for the Flight Dynamics and Design (FDD) software through the adoption of Object-Oriented Software Engineering and leveraged reuse. Some of the reusable software components developed using ROSE will be used in mission and safety-critical applications.

The ROSE Project has chosen to use the Object Modeling Technique (OMT) being developed by James Rumbaugh, et.al. The project will implement the models using Paradigm Plus, which supports Object, Dynamic, and Functional models, and will add an animation capability to the Dynamic model. They will also add constraint checking to all model types to provide consistency and integration checking across the models. These capabilities will be used to simulate the models as a V&V activity.

Along with members of the ROSE project that are participating in domain engineering and in V&V activities, we will be exploring the question of which V&V activities to perform in domain engineering. We will also investigate the impact of domain engineering V&V activities on V&V activities performed during application engineering. These activities may include dependencies on considerations such as the domain area, the reuse methodology, the reuse business strategy archetype, and the domain model representation.

4.3 V&V Tools Under Development

Software tools represent a means for development teams to improve their process through formalization and automation of their activities. By developing new tools or adapting existing tools, a team can formalize their existing practices and incrementally improve their process. We do not advocate the purchase of expensive Computer-Aided Software Engineering (CASE) tools without careful consideration, rather we believe that the act of developing or adapting tools can lead development and IV&V teams to discover aspects of their processes.

By developing analysis tools, IV&V teams can discover many things about the software development and IV&V processes. For instance, a tool developed by CTA on EOSDIS helps keep track of interface definitions between system components. Early in a project these interfaces are abstract and ambiguous. As the system design evolves, the specific details of system component interfaces are refined. The CTA tool helps keep track of the relationships between abstract interface descriptions and specific interface definitions later in the process. Inconsistencies in these relationships can indicate problems and potential sources of misunderstandings in the project design between different development vendors. By developing this tool, the EOSDIS IV&V team has learned much about the process of system architecture development within the EOSDIS development effort.

We have taken a similar approach within the IV&V research effort. By developing V&V tools, we hope to improve the practice of V&V on existing projects and better understand V&V analysis and its relationship to the software development process.

4.4 The Emerging Paradigm

Many questions have been raised regarding the role of V&V within new software development paradigms such as Rapid Development Methods (RDMs). RDMs emphasize rapid development of software through incremental deployment of product functionality. They allow software products to evolve as project requirements change, as a project undergoes budget and schedule pressures, and as hardware platforms evolve. Many companies are now using RDMs to build and maintain software product lines.

What is the role for IV&V in this new paradigm? Such questions are important to ask as software development organizations mature in their enterprises. For V&V, the RDM paradigm means that analysis activities must be tightly coupled with and parallel development activities. Analysis from IV&V must provide immediate or near-immediate feedback into the development process. This feedback must be meaningful and help guide development organizations in their decisions regarding design changes, functional enhancements, and bug fixes.

RDM is similar to software prototyping under the pragmatic view that most prototypes evolve into products anyway. Rapid development implies that the designers of the product are concerned primarily with providing functionality and refining requirements in an operational fashion (aka prototyping). This concern with functionality can lead to poor long-term design decisions and designers can also overlook subtle, complex interactions between functions. In other words, the designers are most concerned with nominal behaviors of the software product.

An IV&V organization plays an essential role in the RDM paradigm. IV&V is concerned with off-nominal behaviors. IV&V tries to find the subtle, complex interactions between functions and components that take some extensive analysis. These are exactly the behaviors that are of little interest to the Design group in the prototyping and RDM models of software development. These two groups, Design and IV&V, can work together in a synergistic fashion to produce high-quality software, on-time, and within budget. One group can focus on functionality while the other is concerned with long-term design goals. These long-term goals are important to provide a conceptual framework within which a product can grow in functionality.

This bipartite approach, called the Rapid Design and Test (RDT) paradigm, is an emerging paradigm in software development organizations. We do not claim that RDT is something new, rather that it is already being practiced widely and has only recently been identified recently. For example, Microsoft Corporation employs an RDT model in developing their products. Their technical staff consists of 1800 "software design engineers" and 1800 "software test engineers" who work in separate, co-equal groups. These groups coordinate their activities through tightly-coupled "daily builds" that force product upgrades based on bug fixes and new functionality. More information on this paradigm will be available in a book by Michael Cusumano and Richard Selby entitled Microsoft Secrets to be released in September 1995 by Prentice-Hall.

An IV&V organization plays the role of a "test" organization in an RDT model throughout the development lifecycle. If we think of any IV&V analysis as a "test" of the current design, IV&V can provide immediate feedback in a synergistic fashion to a development organization. It can look for subtle, complex interactions between system components and functions. IV&V can alert development to their presence as early as possible in the lifecycle. IV&V can fulfill both of its primary objectives within the RDT model: increase quality through immediate, added-value feedback and long-term avoidance of high-cost errors. The RDT paradigm affirms IV&V's value and role especially within large, complex software development projects.

4.5 IV&V Effectiveness

One of the most important questions facing the manager of any software development effort is "How do I build a high-quality software system within my time and budget constraints?" The answer to this question, if one exists, is likely to be complex. As Fred Brooks states, there is "no silver bullet" that will help us build software quickly and easily because software, by its very nature, is inherently complex. Independent verification and validation (IV&V) has been promoted as an effective technique used on the development of large, complex software systems to build high-quality software on time and within budget. But how much IV&V is appropriate and how does a manager determine its effectiveness? Simple answers to these questions are likely to confound the problem and oversimplify the issues.

Strong evidence points to the effectiveness of IV&V on large, complex projects. Extensive reports such as the JPL Study on IV&V Effectiveness, the Leveson Assessment of Shuttle Flight Software and other many other references strongly recommend IV&V on major projects where cost, quality, and schedule problems can have dire consequences. The lack of a continuous, independent technical review of project artifacts and activities can lead to disastrous problems. Furthermore, projects with some form of IV&V practiced from early in the development lifecycle have reported significantly better success rates. Such assessments, however, are anecdotal and quantitative analysis is limited.

We believe that IV&V activities are fundamental to large, complex projects. This currently makes it difficult to clearly identify the benefits of IV&V in a quantitative manner since it is difficult to separate the added-value of IV&V from the rest of the software development effort. For example, issues raised by the IV&V contractor at milestone reviews are not always captured. Likewise, suggested solutions by IV&V are not always recorded during reviews. It is unrealistic to expect that all reviews, suggestions, comments, and communication be recorded in detail regarding IV&V activities because such an effort would put an undue burden on IV&V practitioners.

Furthermore, there is a Hawthorne Effect on software development due to the presence of IV&V on a major project. The mere presence of a group such as IV&V in the process lifecycle has significant effects on the behavior of the software development group itself. It is very difficult to assess the hypothetical effect of the absence of IV&V in such an environment.

All of these points lead us to a preliminary conclusion that simple metrics do not tell the entire story regarding IV&V effectiveness on large, complex projects. This has long been true for measuring other aspects of the software development effort such as programmer productivity, software reliability, estimated errors per line of code, code complexity, and a host of other measures. These measurement problems have been and continue to be the subject of ongoing research. Measuring IV&V effectiveness is no less difficult a problem.

Under the direction of Dr. Ralph Neal at the NASA/WVU Software Research Lab here in Fairmont, we have begun to examine empirical data on major software projects that have employed IV&V. Dr. Neal and Dr. Joshi of Fairmont State College are conducting an analysis of IV&V data on the Day-Of-Launch I-Load Update (DOLILU) software for Space Shuttle Ground Support. This analysis will help form a basis for sound software metrics on IV&V effectiveness.

Current software measurement is crude in the sense that at best it reveals potential strengths and weaknesses in a software development process and its related artifacts. Current techniques do not provide definitive results that provide designs, but rather provide design support to management.

The fundamental relationship between IV&V and the design aspects of software development is very deep and not well understood. In the last issue of the VRQ (January-March 1995), we reported on research on a paradigm that is emerging in software engineering where analysis from IV&V provides immediate or near-immediate feedback to designers throughout the software development process. This feedback helps guide development organizations in their decisions regarding design changes, functional enhancements, and bug fixes. In such a model, the design part of development is concerned primarily with nominal behaviors of the software product. IV&V, however, is concerned with off-nominal behaviors of the software system. IV&V tries to find the subtle, complex interactions between functions and components that take some extensive analysis. These are exactly the behaviors that are of little interest to the design group, especially in prototyping, incremental, evolutionary and other rapid development models of software development. These two groups, design and IV&V, can work together in a synergistic fashion to produce high-quality software, on-time, and within budget. One group can focus on functionality while the other is concerned with long-term design goals. These long-term goals are important to provide a conceptual framework within which a product can grow in functionality.

An IV&V organization plays a vital and fundamental role throughout the development lifecycle. If we think of any IV&V analysis as a "test" of an evolving design, IV&V an provide immediate feedback in a synergistic fashion to a development organization. It can look for subtle, complex interactions between system components and functions, IV&V can alert development to their presence.

Approaches that use the number of issues reports (e.g., RID, DR, CR, DITR, etc.) found by IV&V versus the total number of reports created on a project oversimplify the role of IV&V on a project. Such approaches ignore the nature of the software process relative to the impact and frequency of reporting on requirements, design, and code activities. For example, IV&V may be identifying significant problems, but if synchronization is infrequent between development and IV&V then the benefits of IV&V can be lost. Another example concerns traceability of activities throughout the software development lifecycle. If requirements are not updated and do not trace clearly to design and code, then an outstanding requirements review may have little impact.

Studies have shown that IV&V can be a highly effective approach to software development if it is employed correctly. Any measurement must take these dimensions of the process into account before quantitative statements can be made about the effectiveness of IV&V on large, complex projects. Until research is complete, project managers are ill-advised to adopt unqualified measurements and must examine anecdotal evidence on their own project to determine the benefits of IV&V.

Chapter 5

Metrics

This chapter describes ongoing work in the area of softare metrics in IV&V. We discuss current experiments, tools, and preliminary results of collection, processing, and analysis of software metrics for development and V&V activities.

5.1 Defining Metrics for the ROSE Project

A Memorandum of Understanding (MOU) is being considered among NASA/WVU Software Research Lab (WVU), NASA Johnson Space Center (JSC) NASA Mission Operations Directorate (MOD), and Rockwell Space Operations Company(RSOC) to study the metrics being collected by the ROSE project. The thrust of the study is to identify the smallest set of metrics both necessary and sufficient to measure the important dimensions of the software.

It is not beneficial to measure the same dimension of an object by more than one method. Each method will have its own degree of accuracy and its own cost of application. Once the necessary degree of accuracy has been established, the most cost effective method that delivers that level of accuracy should be the measurement of choice.

This study will:

1. identify important dimensions of the software,

2. classify metrics by the dimension(s) they measure, and

3. use multivariate statistical methods to investigate the parallelism/orthogonality of the captured metrics.

5.2 Misrepresentation of Software Project Statistics

It is widely acknowledged that software projects often over-run on budget and schedule, and in some cases never deliver usable software. But exactly how bad is the problem? When faced with this question, many software engineers cite the GAO study, published in 1979, which stated that only 2% of software contracted for was usable as delivered. The commonly cited figures from that study were:

50% of contracts had cost overruns
60% of contracts had schedule overruns
45% of software contracted could not be used
29% of software was never delivered
19% of software contracted had to be reworked to be used
3% of software contracted had to be modified to be used
2% of software contracted was usable as delivered

These data are correct, but highly misleading. To see why, it is necessary to take a closer look at the study itself. The study examined a number of software contracts, commissioned by various federal agencies, for custom-built business and administrative systems. Two surveys were performed, followed by detailed case studies. The first survey covered 163 contractors, but the results were never published. The second covered 113 Federal data processing personnel with contracting experience. The results of this survey are shown below. Compare the percentage of respondents saying that "software paid for but never used" happens rarely (57.1%) or never (20.5%) with the set of figures above. Surely some mistake?

	Very common	Fairly common	Not very common	Very rare	Never occurs	Don't know
Software development has dollar overrun	21.2	29.2	25.7	9.7	6.2	8.0
Software Development has calendar overrun	30.1	31.9	25.7	8.0	1.8	2.7
Delivered software must be corrected or modified by in-house programmers	8.8	34.5	35.4	13.3	6.2	1.8
Software is paid for but never used	0	3.6	16.1	57.1	20.5	2.7
Delivered software is difficult to modify	5.3	37.2	38.1	11.5	4.4	3.5

The first set of figures above were produced from the second part of the study, a detailed analysis of nine software contracts. Eight of these contracts were chosen for study because they were known to be problematic; several of them had pending litigation at the time. Just one of the nine was chosen as an example of good practice. The nine projects totaled $6.8 million. The one good practice project cost just $119,800, or around 2% of the total cost of the contracts studied. Hence the result "2% of software contracted for was usable as delivered". The nine projects were never intended to be a representative sample; it is only their repeated presentation in the literature that has given them this interpretation. And as we have seen, not only was this not a representative sample, but the figures give a misleading impression of the nine projects studied: 2% (by cost) sounds a lot more dramatic than 11% (or one ninth)!

The real aim of the study was not to determine what proportion of projects fail, but to find out the causes of failure. The table below indicates that the primary causes were contractual and management, with failure to specify requirements adequately occurring on all but one of the eight problematic contracts. (The "good practice" case is easy to spot in the table).

Cause	Case Number								
	1	2	3	4	5	6	7	8	9
Agency overestimated its own state of progress when it let the contract	x	x				x			x
Incorrect agency management action, such as using inappropriate contract			x	x		x		x	
Agency failed to specify requirements adequately	x	x	x	x	x				
Agency over-committed itself	x	x				x	x		x
Agency failed to manage during execution, including excessive changes	x	x				x	x		x
Agency failed to adequately inspect and test	x	x	x			x		x	

There are a number of important lessons to be learned here. The first is the danger of misrepresenting quantitative results from software projects. The trend towards more measurement of the software process, and the drive for empirical validation of process improvements, should increase our knowledge of the software process. But there is also a danger that results will be misinterpreted, leading to ill-advised decisions. For example, there have been a number of attempts to measure the cost effectiveness of IV&V. Cost effectiveness here usually means "is the extra cost of an IV&V contract recouped in savings from fewer errors and reduced maintenance costs?". Suggestions for calculating this typically involve determining the cost to detect and correct various types of error on projects that do not use IV&V, as a way of estimating the savings from early detection of similar errors on projects that have IV&V.

Most such studies have concluded that IV&V is cost effective. Figures for the benefit to cost ratio for IV&V have been as high as 5 to 1. A simple interpretation of these figures might be that for every $1 spent on IV&V, $5 are saved in fixing errors. Of course, we shall avoid such simple interpretations, as this ratio is highly dependent on the type of project, and the way in which IV&V is applied. There is also a law of diminishing returns for IV&V spending, with the optimal figure generally reckoned to be around 12-18% of the development cost, depending on the size of the project.2

One of NASA's studies, however, indicated that IV&V was not cost effective. In 1982, Goddard's Software Engineering Laboratory (SEL) found that introducing IV&V resulted in a huge overall productivity decrease, representing an increase in cost of around 85% over similar software products. At the same time, there was no significant benefit in any of the areas measured except for an 84 to 90% decrease in requirements ambiguities and misinterpretations. They found no significant decrease in the number of design errors detected, and no significant decrease in the cost to fix errors. This led them to conclude that IV&V was inappropriate for their environment, a conclusion that still appears in SEL reports, thirteen years later.

Just as it proved important to question the interpretation of the GAO results, so the interpretation of the SEL's study should be questioned. Firstly, it is not clear the technique used in the SEL study really was IV&V. The "IV&V" team was a part of the same contractor as the development team, and certainly did not have managerial independence. Secondly, the IV&V team had to use the same (limited) resources as the developers, and competition for these resources appears to have worked to the detriment of both teams. In fact, the projects studied were a part of an operations environment, rather than a development environment, in which developers already had a low priority for computing resources. The addition of an IV&V team merely exacerbated this competition. Finally, the measures used in the study only cover the development phase, not the whole lifecycle, so savings from improved maintainability were not measured. Hence, we should conclude that not only is the applicability of the result to other projects severely limited, but that the results themselves are in question. Even the participants of the study have voiced their concerns over the interpretation of the results.

To conclude, empirical results from software projects have to be examined very carefully before being accepted by the software engineering community. We have criticized the interpretation of two studies, both of which are more than ten years old. However, in both cases it is far more common to find the studies cited as representative results than it is to find them questioned.

Finally, note that the type of empirical work we have described depends very much on the ability to compare projects. For example, the cost effectiveness of a particular technique can only truly be calculated by isolating it as the only variable factor over two or more projects. At the very least, this means comparing two or more projects of similar size, application domain and development environment, one of which used the technique and one of which did not. Hence, evaluating the cost effectiveness of IV&V on a program such as the Space Station is impossible, as there is no other comparable project anywhere. More importantly, if there were a comparable project, it is unlikely that such a project would risk not having IV&V!

[1] Blum, B. (1992) "Software Engineering: A Holistic Approach", Oxford University Press.

[2] Lewis, R. O. (1992) "Independent Verification and Validation: A Lifecycle Engineering Process for Quality Software", New York: Wiley.

5.3 Managing Software Projects Using the WWW

One important aspect of software development and IV&V is measurement. Unless a software development effort can be measured in some way, it is difficult to judge the effectiveness of current efforts and predict future performance. The Web-Integrated Software-metrics Environment (WISE) is an environment that provides software projects with the capability to develop applications that allow project managers and developers to coordinate project activities and collect metrics.

Collection of metrics and adherence to a process are difficult tasks in a software project. Watts Humphries in his book Managing the Software Process states:

> The most disastrous mistakes are often made when the project is under the greatest schedule pressure. These are often caused by a loss of change control that started with a quick object patch. While the harried programmers invariably intended to document their changes whenever there was time, it is extremely difficult to remember precisely what was done and why.

Automated systems that handle change requests, issues, problem reports, activity log entries, and other process documents provide an excellent platform for tracking the status of the project. The WISE project management tool helps developers keep track of problems, maintain record of changes, solved problems, issues, and errors. WISE helps developers coordinate the work on a project, distribute effort, track of discrepancies, log of all entries, and maintain a continuous flow of data between the various groups.

WISE began as an idea to put a programmer's "to-do" list on the WWW and allow programmers to view their own metrics as well as group metrics. WISE also tracks "to-do" items between members of a group. For example, an error is logged initially as an open issue (i.e., a "to-do" item). Other developers on the project see this new open issue appear on their "to-do" lists. A developer will mark the issue as fixed when the problem has been rectified. Fixed issues show up on a test engineers "to-do" list who will mark the issue as closed when successfully retested.

WISE also serves as a performance indicator. WISE can track when changes to an issue occur and provide project metrics in embedded graphics (i.e., plots, pie charts, etc.) within WWW pages. We have created several views of group and individual metrics on projects. Individuals can control access to their data by others and group views provide only aggregate information.

5.3.1 An Overview of WISE

The Web Integrated Software Environment (WISE) is a system for managing the dynamics within a software development effort. Access to documents and the life cycle of issue reports can be managed by standard WWW browsers (e.g., Mosaic, Netscape, etc.). Web-browsers and other user programs that can access the WWW can be used for this automating effort. Some advantages that such an automated effort brings to a development group are:

- WISE overcomes the geographical barrier. By using a WWW client one can access another resource anywhere on the Internet. This benefits the software teams who can access the tool from any place.
- WISE overcomes the communication barrier. Encourages collaborative software problem solving.
- WISE facilitates problem solving. Software managers can effectively track the progress of their work group by using performance measures built into the tool itself. WISE supports different views the tool supports of a problem database and helps the managers handle the large amount of detail and help them isolate problems early and suggest timely action
- WISE provides data continuity. Another advantage is that changes to the database are reflected in all views. There is a flow of data from one to another and every update is kept track of. This kind of continuous flow of data from the user to the managers and back, helps coordinate the work products of many different people who work on common projects.

5.3.2 A Brief Description of WISE

WISE is a WWW-based issue tracking tool. The clients run a browser program and a WWW server answers client requests. With the vast variety of browser programs for UNIX, Macintosh, and Windows, one can access the WISE with great ease.

The user of the tool specifies the URL (Uniform Resource locator) of the information server, logs into WISE, and views their personal "to-do" list. They may perform one of the many kinds of operations like logging a new issue or acting on existing issues. They may also view their individual or group metrics. Some group metrics are also available to the general public.

The current backend of WISE is an Oracle database, but this is being extended to work with other SQL-based engines. Every time a user selects an option, a query containing the user's request is sent to the backend through a gateway that helps connect to the underlying database. The information server returns with the data and the user views them. To solve the gateway problem from the WWW server to the Oracle backend, we initially decided to use an in-house software called Web* which had an Orbix interface to Oracle embedded in Tcl/Tk. We implemented a prototype tool using this software but were not very comfortable with the backend and the dependencies it attaches to the tool. So we decided to search the Web for better tools.

While surfing through the net we came across a gateway program that provides a form interface in Mosaic to SQL databases. GSQL-ORACLE is a slightly modified version of the same software except that it is specialized for Oracle SQL backends. The working of the tool is as follows: GSQL parses the forms based on some specification files we write and creates SQL statements. Then GSQL invokes the oracle backend with the SQL query and the arguments filled in by the user. The backend processes the query and returns HTML documents. These results are sent to the client running one of the standard browsers.

5.3.3 Current Status of WISE

We now have a prototype tool working. We feel that it is very important to use this tool in our own efforts. Indeed, we are using WISE to manage WISE's development. The current version of the WISE tool uses GSQL as a backend to our Oracle gateway (a DEC machine running RISC-ULTRIX 4.3). Many different browsers can access WISE through the World-Wide-Web. There are a limited set of metrics available, but these are growing rapidly as we see the need for them in our projects.

WISE also provides metrics on projects in the form of graphs that compare aspects of the group activities. The system can answer queries about the evolution of issue reports in the system. For example, a manager can query the number of open vs. closed issue reports over time in graph form. The performance of individuals are also available but only to each individual. WISE has strict boundaries on the availability of metric data on individual performance. WISE allows individuals control over the visibility of their metrics to managers and others.

WISE is non-intrusive because it provides a status list to each developer in the team. Each element in the list can be acted upon which will change the element's status. This might make it show up on another developer's status list.

We are also working on a programming language to specify WISE forms. The current system is "hard-wired" in CGI scripts, but the Wise Programming Language (WPL) is a software process language that allows form views to be specified within the software process. A WPL specification is a collection of form definitions and views. The composition of the forms and views defines the totality of the software process. Thus, the process is not fixed or globally defined by a manager, but is dynamic and changes based on the roles of development personnel. The composed process can be checked for consistency to ensure that invariant and liveness properties are conserved.

The WISE project (Jack Callahan, Sudhakar Ramakrishnan & Wei Sun) would like to thank Jason Likkai at NCSA and James Pitkow at Georgia Tech for their help. We also thank a number of people at NASA IV&V facility at Fairmont and CERC. A demo of WISE is available at http://research.ivv.nasa.gov/.

Interest in our WWW Integrated Software Environment tool, called WISE, has been tremendous! We have received correspondence from over 100 sites world-wide asking for the software and over 10 sites submitted proposals as part of an Alpha test plan. These sites include Sun Microsystems, IBM, Salomon Brothers, and Cnet Online. To find out more about WISE, please look at our research WWW home page at http://research.ivv.nasa.gov/. You will find a brief demo of WISE at this site.

The Web Integrated Software Environment (WISE) is a system for managing issue reports within a software development effort. Access to documents and the life cycle of issue reports can be managed by standard WWW browsers (e.g., Mosaic, Netscape, etc.). Web-browsers and other user programs that can access the WWW can be used for this automating effort.

WISE began as an idea to put a programmers "to-do" list on the WWW and allow programmer to view their own and group metrics. WISE also tracks "to-do" items between members of a group. For example, an error is logged initially as an open issue (i.e., a "to-do" item). Other developers on the project see this new open issue appear on their "to-do" lists. A developer will mark the issue as fixed when the problem has been rectified. Fixed issues show up on a test engineers "to-do" list who will mark the issue as closed when successfully retested. WISE also serves as a performance indicator. WISE can track when changes to the issue occurred an provide project metrics in embedded graphics (i.e., plots, pie charts, etc.) within WWW pages. We have created several views of group and individual metrics on projects. Individuals can control access to their data by others and group views provide only aggregate information.

We are also working on a programming language to specify WISE issue forms. The current system is "hard-wired" in CGI scripts, but the Wise Programming Language (WPL) is a software process language that allows form views to be specified within the software process. A WPL specification is a collection of form definitions and views. The composition of the forms and views defines the totality of the software process. Thus, the process is not fixed or globally defined by a manager, but is dynamic and changes based on the roles of development personnel. The composed process can be checked for consistency to ensure that invariant and liveness properties are conserved.

Chapter 6

Testing

Testing remains one of the most practiced approaches to sofwtare quality assurance. We outline research that is exploring new testing techniques in this chapter and the next chapter (Classification).

6.1 Validation Testing – An Automated Approach

The test approach defined within this white paper can be tailored to various levels of testing (i.e. unit level, integration, or system certification/validation). However, the focus of this paper primarily addresses system certification/validation testing. The intent of this test approach is to provide mechanisms by which validation of a system can be easily confirmed relative to the execution of a test suite. The basis for this approach relies heavily on the establishment of a requirements database and the allocation of requirements to functional test threads, test cases, and test procedures. The automation associated with the approach requires capturing functional test thread, test case, test procedure and the recorded test session result information in a relational database management system (RDBMS). Once a process for capturing this information in a RDBMS is adopted, a determination of the requirements satisfied by successful test steps/procedures is deterministic. Structure Query Language (SQL) queries can be issued to the test database to determine certification status at any given point in time, thereby exploiting the power of SQL so that management indicator reports can be generated during certification testing activities.

The automated/database oriented approach advocated yields the greatest benefit and is essential to validation testing for large systems. When integrating or validating large systems, the amount of information that must be managed is too great to address by the traditional hard copy document approach. In addition to the obvious information management benefits, the following benefits may be realized by utilizing this automated database driven test approach:

- provides a basis for automating test resource scheduling (i.e. hardware, software, etc.)
- allows management real-time access and insight into test activities to determine performance against schedule
- supports regression testing and allows for variance analysis between testing activities
- enhances information sharing among the test team
- establishes a foundation for dumping database data to deliver hard copy plans, procedures, and test reports
- provides a forum for recording test discrepancies in support of discrepancy reporting
- automates the collection of test metrics allowing for metrics reports to be generated based upon database queries

6.1.1 Definitions

1. A functional test thread as referred to in this paper coincides to an end-to-end path through the system. In defining a thread, the inputs and expected outputs must be identified. Note that a thread may span several system components, thus a single functional test thread may be associated with one or many test cases.
2. Refer to the MIL-STD-498 documentation standard for a definition of a test case and the information that should be maintained for each test case. Note that there is a one-to-one correspondence between test cases and test procedures.
3. Refer to the MIL-STD-498 documentation standard for a definition of a test procedure and the information that should be maintained for each test procedure.

6.1.2 Recommendation

1. Develop a client/server application where test thread, test case, test procedure and test result information can be input on the platform most used. Several Rapid Application Development (RAD) tools exist which create client applications for PC, Mac, and Unix platforms. Consider Powerbuilder from Sybase (formerly a Powersoft product), SQLWindows from Gupta Corporation, and JAM from JYACC Corporation. The benefits associated with building a client/server application include sharing of data between geographically dispersed individuals or teams and support for either centralized or distributed databases.

2. Select an RDBMS sufficient to support the resource requirements for the project. Criteria should include number of users, amount of data, etc.

6.1.3 Implementation

A Test Management Database (TMDB) application was built for the Earth Observing System Data and Information System (EOSDIS) Core System (ECS) integration and system certification test activities. The MIL-STD-498 was utilized to define much of the information supported by the TMDB schema. The TMDB is a client/server application that was built using the Gupta SQLWindows product. The TMDB is a PC/Windows client application which communicates to an RTM/Oracle database and a Sybase SQL Server database, both of which reside on a Sun SPARC 1000 running Solaris. The Requirements Traceability Management (RTM) COTS tool from Marconi serves as the requirements repository delivered by the developer. RTM uses Oracle as the RDBMS where it stores data. All test thread, test case, test description, and test result information is stored in the Sybase SQL Server RDBMS. Gupta Corporation is in beta test of a Unix client and is expected to support Mac clients in the second quarter of 1996.

6.2 Requirements-based Testing and Analysis

How should we represent requirements for a large software system? Many projects still rely on natural language as the main way of representing requirements. And yet for complex requirements, natural language is notoriously ambiguous. This article describes a simple experiment to demonstrate the ambiguity of English language requirements, and discusses the benefit of using redundancy to improve detection of ambiguity.

Consider this requirement, from the FDIR (Fault Detection, Isolation and Recovery) requirements for Space Station:

> (2.16.3.f) While acting as the bus controller, the C&C MDM CSCI shall set the e,c,w, indicator identified in Table 3.2.16-II for the corresponding RT to "failed" and set the failure status to "failed" for all RT's on the bus upon detection of transaction errors of selected messages to RTs whose 1553 FDIR is not inhibited in two consecutive processing frames within 100 millisec of detection of the second transaction error if; a backup BC is available, the BC has been switched in the last 20 sec, the SPD card reset capability is inhibited, or the SPD card has been reset in the last 10 major (10-second) frames, and either:
>
> 1. the transaction errors are from multiple RT's, the current channel has been reset within the last major frame, or

2. the transaction errors are from multiple RT's, the bus channel's reset capability is inhibited, and the current channel has not been reset within the last major frame.

We gave this requirement to four different people, and asked them to produce a truth table to clarify the conditions under which the required function should be provided. We received four different answers, which differed in both the number of conditions identified (i.e. number of rows in the table) and the number of combinations under which the function would be activated (i.e. columns in the table).

The differences in the responses show that the original requirements statement is riddled with potential ambiguities. For example, the mixture of "ands" and "ors" the requirement is a problem because, unlike programming languages, English does not have any standard precedence rules. It is not clear how to scope the various subclauses, either. For example, the timing condition within 100 millisec... could refer to the inhibition of the FDIR, or to one or both of the required setting operations. With some domain knowledge, it is possible to guess the most likely interpretation, but this is by no means a trivial task, and there is no guarantee that everyone who needs to read this requirement will get it right.

Ideally, we would like to ensure that everyone who reads a requirement will interpret it in exactly the same way, the way that was intended. As with any communication act there is a balance of responsibility between the author and the reader: the author must take steps to ensure her words are not open to misinterpretation, and the reader should take steps to interpret carefully the meaning of the words. The author cannot be expected to anticipate all stupid or disingenuous interpretations, but at the same time is under an obligation to ensure that the meaning is clear for the intended audience.

One could argue that our experiment was unfair, as the requirement was taken in isolation, and the people we gave it to didn't have the domain knowledge to help interpret it. Furthermore one could argue that the designers will know how to interpret it correctly, through interaction with the team responsible for writing it. Unfortunately, these answers undermine the whole point of writing down requirements: it is important to capture all the relevant information so that a wide range of people (implementors, testers, maintainers, users, technical authors, etc) can access it over a long period of time.

So how can we reduce the chance of misunderstandings occurring? A quick and easy solution is to exploit redundancy to help detect misunderstandings. Consider this simple function specification:

This function shall compute the square root of its input, to within three places of decimal, i.e. for input x, it will output y such that x $\leq$ y ¡ x+0.001

Strictly speaking this specification is redundant: either the English language part, or the mathematical definition could be missed out. However, the redundancy acts as a double check that we have understood the requirement. If we read both parts of the specification, and they seem to say the same thing, we can be much more sure that we have understood what the author intended. If the two parts seem to be saying different things, then either we have misread one of them, or the author made a mistake.

In the example above, the two parts do not say the same thing. The mathematical description says that the error can only be positive, i.e. that the value returned is never less than the actual root. The English description does not include this restriction. Such a restriction would be very important to the implementor, because most square root algorithms cannot guarantee a positive error. Having spotted this, we can now go back and check which version is correct. If we had only been given one part, we would not have thought to question this issue.

We can adopt a similar approach throughout a specification. For example, where a requirement like the FDIR requirement given above appears, the English description could be accompanied by the tabular form. Again, if the two parts are consistent, then we can be fairly confident that we have understood the requirement. The implications for IV&V should by now be clear: redundancy provides the IV&V team with an powerful means of spotting errors in the expression of the requirements.

6.3 The SRL Testing Toolkit

The SRL Testing Toolkit will be a toolbox of useful testing tools aimed at providing mechanisms for programmers to develop code that closely meets system specifications. The toolkit will contain tools that are continually being used to perform testing on RMP. In addition, these tools are undergoing a transformation and growth so that they can accommodate other complex pieces of software.

The main goal of the toolkit is to provide techniques and applications to help developers structure their tests better and to get more tangible benefits out of the testing process. Our view of testing is that it is ultimately essential in any and all systems. However, the ways to approach it are lined with possible pitfalls. The toolkit will attempt to guide developers and testers down the paths that we have discovered lead to good payback.

The toolkit is separated into three main categories. These are:

- Test Visualization and Generation tools,
- Source Code tools, and
- Test Execution and Analysis tools.

The Test Visualization and Generation tools will include tools that are directed at helping testers and developers visualize and generate test cases for their systems. The Source Code tools will be a set of tools geared towards allowing the developer and tester place statements in the code in order to see what state the system operates through. This technique is very simple and extremely useful. All developers place print statements through their code, why? Typically, the answers are such things as: "using the debugger is too time consuming", "the system can not be stopped, it must process without interruption", or "the debugger is overkill for this problem". All of these answers stem from the fact that most debugging environments have not kept up with the advances in software over the last 10 years. Advances such as multi-threading, continually running network servers, and highly complex distributed systems are impossible to debug using conventional debuggers. In addition, these debuggers change the layout of the executable in ways that actually make some problems seen during operation go away while debugging. Clearly something lightweight and as unobtrusive as possible is needed. This is the domain of the humble print statement that programmers have used for decades. The toolkit takes these statements to the next level by providing mechanisms to capture them and analyze them in real time.

We plan for the toolkit to support these platforms and languages:
- Windows 3.1, NT, and 95,
- BSD and SVR4 derivative UNIX systems, and
- C, C++, Tcl, Tk, Python, Perl, and Java.

As a first step, the toolkit will be released incrementally. The first piece will be the logging facility scheduled for release in January. Other pieces of the toolkit will be released as they become generally stable and usable.

Chapter 7

Classification

Because exhaustive testing is impossible on large systems, analysis of all sorts (including testing) must be prioritized. This means that analysis must classify parts of a software development project. Both process and products must be classified for analysis in order to perform a cost-effective analysis of a software development effort.

The work on classification has focused on the use of formal methods to structure, manage, and generate test cases. The formal approach allows us to make rational decisions regarding which functions and components of the system are critical based on a formal statement of system requirements. We have used our work on the Reliable Multicast Protocol (RMP) as a testbed to explore the viability, strengths and weaknesses of this approach to classification. The following sections discuss our work related to new classification techniques on the RMP project.

7.1 Verification and Validation of RMP

Much work has been done in the area of verifying that implementations of communication protocols conform to their specifications. Conformance is usually verified through extensive testing of an implementation in which tests are derived directly from the protocol specification. If an implementation behaves in a manner predicted by the protocol specification, then the implementation is said to conform to the specification. If not, then an error exists in the implementation of the protocol. Although this method does not formally verify that a protocol specification and an implementation are consistent, it represents the state-of-the-practice in this domain of software development.

In the recent paper[1] we describe our experiences while trying to formally

[1] J. Callahan and T. Montgomery, An Approach to Verification and Validation of a Reliable Multicast Protocol, in *Proceedings of the ACM International Symposium on Software Testing*

specify and implement a complex communications protocol that provides reliable delivery of data in multicast-capable, packet-switching telecommunications networks. The protocol specification, called the Reliable Multicasting Protocol (RMP), was developed concurrently with its implementation. The implementation was developed incrementally using a combination of formal and informal techniques in an attempt to ensure the correctness of its implementation with respect to the evolving protocol specification. We found that many formal methods did not help us in the development of the protocol specification nor its implementation. We concluded that the best uses for formal methods in our situation was in the specification of the protocol requirements and the generation of tests derived from the specifications applied to prototype versions of the software during development.

One of the primary goals of our effort was to achieve high-fidelity between the specification and implementation during development. High-fidelity means that the specification model and implementation agree regarding the behavior of the protocol. We felt that if fidelity was not a primary concern, then there existed the strong possibility that the specification and the implementation would diverge in behavior. This would render analysis of any formal specification model irrelevant in the development and maintenance of the software since such analysis would offer little assurance that the actual code behaved in an identical manner.

Our development process involved two teams: a design team and a verification and validation (V&V) team. These two teams worked in an iterative, interactive fashion that allowed the design team to focus on nominal behaviors of the software while the V&V team examined off-nominal behaviors. The task of the design team was (1) to specify the protocol in terms of mode tables and (2) implement the protocol in C++ as specified by the mode tables. The task of the V&V team was to (1) analyze the consistency and completeness of the mode tables by analyzing "paths" through the mode tables and (2) generate tests from the mode tables for suspect conditions. Suspect conditions include those paths identified in the mode table model as being deadlock, livelock, or potential sources of unexpected behaviors. The V&V team used the requirements mode model to identify cases that were considered by the design team to be unusual or virtually impossible. In retrospect, these cases were the source of several errors in the specification and implementation of RMP.

We use the terms "verification and validation" in a different context from their typical usage because of our bipartite, prototyping development process. In our case, the term "verification" refers to activities that help in the identification of off-nominal behaviors of the software based on analysis of the specification model. We use the term "validation" to refer to activities that involve testing the implementation for properties based on potential problems revealed through verification analysis.

---

and Analysis (ISSTA), San Diego, CA, January 8-10, 1996.

The protocol specification as expressed in the mode tables helped us organize and structure tests while developing implementation prototypes. Testing formed the dialogue by which the two teams communicated about the intended behavior of the protocol and its implementation. This paper relates our experiences in developing our approach and describes details of our model-based testing methods. We do not claim to have "formally verified and validated" the RMP specification and its implementation, but rather we have developed a strategy and process by which the evolution of RMP is enhanced by testing and verification. Our approach has been to study the problems that have occurred during development, testing, and operation of RMP. Through a post-mortem analysis of problems, we are trying to find methods that may have discovered problems earlier in the development lifecycle.

We do not claim that RMP has been "verified and validated" to the extent that it is totally correct, rather that we have developed a technique that strengthens analysis and testing in the long-term development of our software. Short term problems did occur, but they helped evolve a specification model in high-fidelity with an implementation. Co-evolution of the formal specification model and the implementation was the most useful result of our study. Our technique allowed our two teams to structure their tests and other analysis activities. Their activities supported each other in the development of the implementation and refinement of the specifications. We will continue to use RMP as a testbed problem and explore new specification and analysis techniques that complement incremental software development activities. We are continuing to evolve the specifications even though the software has been released in an Beta test version. This type of release scheme limits the use of RMP to non-critical projects and helps use explore operational problems. When a problem in operation does occur, we are using the mode tables to trace where the problem occurred. This has been useful in understanding problems, finding why problems were or were not detected earlier, and refining the specification incrementally.

Additional information about RMP can be found at our WWW home page at http://research.ivv.nasa.gov/

7.2 Status Update for RMP

The Reliable Multicasting Protocol (RMP) was used last month during the International WWW Conference to multicast HTML pages around the world on the Internet. RMP, developed by the NASA/WVU Research Project in Fairmont, West Virginia, is used in the X-Web-Teach (XWT) tool developed by the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champagne. Using RMP, the XWT tool can coordinate the simultaneous navigation of HTML pages between thousands of users. When one person follows a hyperlink, then all participants will also jump to the same linked page. RMP is being formally verified and validated as a pilot project in

Fairmont. It is also being used within the NASA EOSDIS architecture being developed by Hughes Applied Information Systems. Details about RMP can be found in the VRQ Winter 94 issue. Papers on RMP, the software itself, and its applications can be found at http://research.ivv.nasa.gov/.

There have been remarkable improvements in the Reliable Multicast Protocol (RMP) development in the last few months. In addition to the formal verification and validation activities that are progressing well, the development team has been able to address Security and Authentication, to provide a new look and feel through an enhanced Application Programming Interface(API), and to finish the first stage of a MS Windows NT and Windows95 version.

The testing techniques used and enhanced by the RMP development and testing teams are now starting to be used in defining a generic testing toolkit .This toolkit will be aimed at helping other developers formally test and analyze other object-oriented systems. More information on this toolkit will be forthcoming as research on developing it progresses.

The RMP development team has addressed the heated issue of security and authentication in networking by placing hooks into RMP operation that allow applications to define and use their own methods. The approach taken is the same endorsed by IPv6, or IPng, where the security and authentication algorithms are optional and orthogonal parts of the protocol operation. RMP does not mandate a particular algorithm for encryption or authentication. It simply allows the application developer to do the actual encryption/decryption and authentication procedure itself, thus allowing great range of choice with respect to what schemes can be employed. An RMP incremental release in late August is the release that supports security and authentication.

With any piece of software meant to be used by a wide range of users, especially software designed for use by other developers, changes are necessary so that the software maintains its applicability and continues to stay usable. After over one year of use, first at several Alpha sites and now a widespread use, RMP has matured to the point that the old API is just not sufficient for the needs of the users. In an effort to attack this, the development team has taken comments from as many users as it can in order to evaluate what a new API needs. The first draft of this API is available on the RMP Home Page (http://research.ivv.nasa.gov/projects/RMP/MRP.html). This new API is to be first deployed in the RMP 1.3 Beta release tentatively scheduled for the first week of October.

Networking software is becoming very widespread on PCs equipped with the Microsoft Windows operating system. The RMP development team has received several requests dating back to February 1995 for an RMP version running on MS Windows. After a little bit of thought and a lot of working with the specific platform, I am pleased to announce that a version of RMP is now working on MS Windows NT 3.5 and Windows95! The first release to support this will be the RMP 1.3 Beta release.