NASA-CR-190716

Configuration Management and Software Measurement in the Ground Systems Development Environment (GSDE)

Victor E. Church
D. Long
Ray Hartenstein
Computer Sciences Corporation

Alfredo Perez-Davila
University of Houston-Clear Lake

February 1992

(NASA-CR-190716) CONFIGURATION MANAGEMENT AND SOFTWARE MEASUREMENT IN THE GROUND SYSTEMS DEVELOPMENT ENVIRONMENT (GSDE) (Research Inst. for Computing and Information Systems) 20 p

N92-31787

Unclas

G3/61 0116930

Cooperative Agreement NCC 9-16
Research Activity No. SE.34

NASA Johnson Space Center
Mission Operations Directorate
Space Station Ground Systems Division

Research Institute for Computing and Information Systems
University of Houston-Clear Lake

TECHNICAL REPORT

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

Configuration Management and Software Measurement in the Ground Systems Development Environment (GSDE)

RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Computer Sciences Corporation in cooperation with the University of Houston-Clear Lake. The members of the research team were: Victor E. Church and D. Long from CSC and Alfredo Perez-Davila from UHCL. Mr. Robert E. Coady was CSC program manager for this project during the initial phase. Later, Mr. Ray Hartenstein assumed the role of CSC program manager. Dr. Perez-Davila also served as RICIS research coordinator.

Funding was provided by the Mission Operations Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA research coordinator for this activity was Thomas G. Price of the ADPE and Support Systems Office, Space Station Ground Systems Division, Mission Operations Directorate, NASA/JSC.

Configuration Management and Software Measurement in the Ground Systems Development Environment (GSDE)

Prepared for

Lyndon B. Johnson Space Center
National Aeronautics and Space Administration
Houston, Texas

by

Computer Sciences Corporation
System Sciences Division
Beltsville, Maryland and League City, Texas

and

The University of Houston - Clear Lake
Research Institute for Computers and Information Sciences
Clear Lake, Texas

under

Subcontract No. 075
RICIS Research Activity No. SE-34
NASA Cooperative Agreement NCC 9-16

February 1992

Preparation:	V. Church
Quality Assurance:	D. Long
Approval:	R. Hartenstein
	A. Perez-Davila

Abstract

This report describes a set of functional requirements for software configuration management (CM) and metrics reporting for Space Station Freedom ground systems software. This report is one of a series from a study of the interfaces among the Ground Systems Development Environment (GSDE), the development systems for the Space Station Training Facility (SSTF) and the Space Station Control Center (SSCC), and the target systems for SSCC and SSTF.

The focus of this report is on the CM of software following delivery to NASA, and on the software metrics that relate to the quality and maintainability of the delivered software. The CM and metrics requirements address specific problems that occur in large-scale software development. This report describes mechanisms to assist in the continuing improvement of Mission Operations software development.

Table of Contents

List of Figures

Section 1 - Introduction

As part of the Space Station Freedom Program, the Mission Operations Directorate (MOD) at JSC[*] is developing a Space Station Training Facility (SSTF) and a Space Station Control Center (SSCC). The software components of these systems will be developed in a collection of computer systems called the Ground Systems Development Environment (GSDE). The GSDE will make use of tools and procedures developed by the SSFP SSE contractor. Both the SSTF and the SSCC will be developed using elements of the GSDE.

During development, SSTF and SSCC software will be configuration-managed using contractor-specified tools in their respective software production environments (SPEs). When the software is delivered to NASA (or sooner, depending on contractor-dependent integration procedures), it will be placed under formal CM on the Ground Systems/SPF (GS/SPF) using the tools provided by the SSE. Integration testing and build-up to delivery to operations will involve both contractor and SSE CM capabilities.

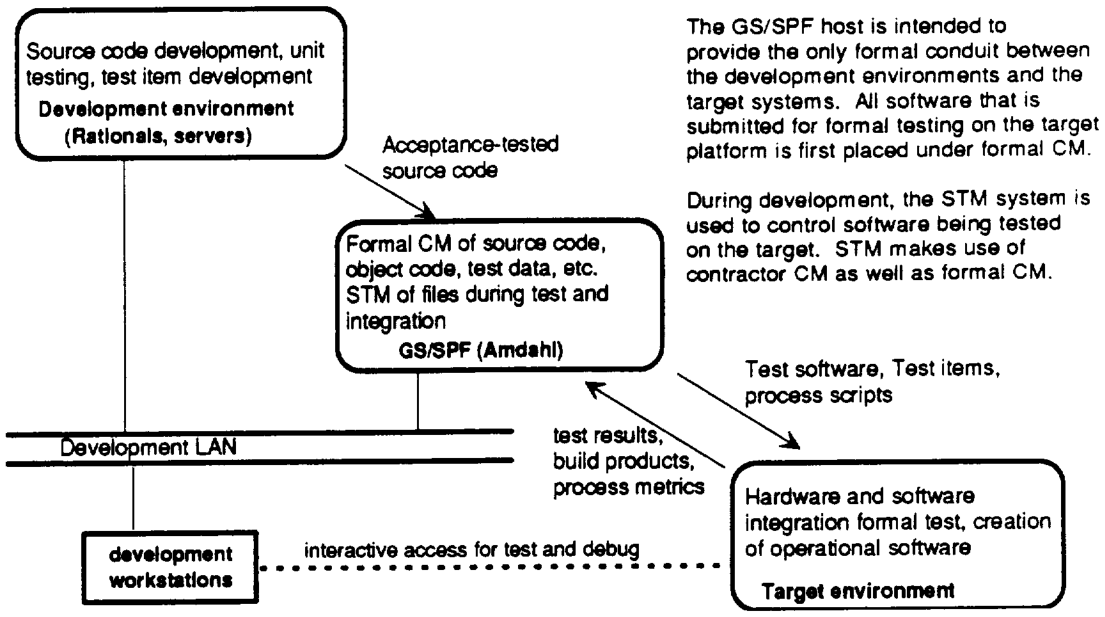
This report specifies basic requirements for configuration management in the integration and test stage of development. An earlier report, GSDE Interface Requirements Analysis, presented overall CM requirements for the use of the GS/SPF. This report extends that analysis based on further analysis and information on the SSCC and SSTF projects. The software measurement requirements presented here are complementary to the CM requirements. They provide additional guidance for software development practices that can improve the quality and maintainability of ground system software.

1.1. Software Development Context

Ground systems software, specifically the SSCC and SSTF, will be developed and tested using combinations of development computers and workstations (collectively referred to as software production environments, or SPEs), a Ground Systems SPF (GS/SPF), and target platforms that are essentially the operational target environments or equivalent. Configuration management will take place in all three environments as software progresses from code and unit test through integration to operational use. Figure 1 shows the basic development context. The target environments include IBM mainframe computers, Unix-based workstations, and mission-specific special-purpose hardware.

This report focuses on how the different CM tools (including the Software Test Management capability on the GS/SPF) will be used to ensure the integrity of software that is delivered to operations. (CM of operational software is outside the scope of this report.)

[*] Acronyms and abbreviations that are in common use in the Space Station Freedom community are not spelled out in the text, but are defined in the Glossary at the end of this report.



Source code development, unit testing, test item development
Development environment
(Rationals, servers)
Acceptance-tested source code
Formal CM of source code, object code, test data, etc. STM of files during test and integration
GS/SPF (Amdahl)
The GS/SPF host is intended to provide the only formal conduit between the development environments and the target systems. All software that is submitted for formal testing on the target platform is first placed under formal CM.
During development, the STM system is used to control software being tested on the target. STM makes use of contractor CM as well as formal CM.
Test software, Test items, process scripts
test results, build products, process metrics
Development LAN
development workstations
interactive access for test and debug
Hardware and software integration formal test, creation of operational software
Target environment

Figure 1. Ground Software Development Environment

Software for the SSTF and SSCC will be developed in the SPEs or in subcontractor facilities, and accepted into the GSDE (at Acceptance Test, or AT) for systems integration and testing. Following AT, the software is placed under formal CM on the GSDE host. The integration and test will take place in the target envirnonment, or on platforms that are essentially equivalent to the target. Following this systems integration test phase, qualification testing will be performed prior to delivery to operations.

The primary language for development of Space Station Freedom software is Ada*. The SPEs will include Ada-compilation platforms (e.g., Rational R1000 computers) to support Ada development. It is also probable that a substantial amount of non-Ada code, primarily C-language, will be developed (or reused) and supported. Workstations and file servers will be used along with the Ada compilation platforms to support development. Configuration management and software measurement in the SPEs will be performed by the developers, using contractor-specified tools.

The GS/SPF is an IBM-compatible mainframe with an instance of the SSE SPF software for each major system (SSCC and SSTF). The GS/SPF will host both the formal CM system and the STM system, along with disk storage supporting both systems. Formal CM will be used to manage software following Acceptance Test. For reasons of security and software integrity, the GS/SPF will serve as the conduit for moving software from the SPEs to the targets.

* Ada is a trademark of the U. S. Department of Defense, Ada Joint Program Office

There are no planned interactions between the two ground system development efforts (SSCC and SSTF). However, there are a number of common interfaces that will be implemented to support the interfaces between SPE and target.

1.2 Organization

Following this introduction, Section 2 presents the CM requirements, including the goals to be achieved through these CM mechanisms and recommendations on how to put those mechanisms into effect. Section 3 describes the software measurement requirements, and places them in context with the DA3 Metrics initiative.

1.3 References and related documentation

Babich, Wayne, *Software Configuration Management: Coordination for Team Productivity*, Addison-Wesley, 1986

Card, David, and Glass, Robert, *Measuring Software Design Quality*, Prentice Hall, 1990

Computer Sciences Corporation, *Digital Systems Development Methodology*, May 1990

Computer Sciences Corporation, *Ground Systems Development Environment (GSDE) Interface Requirements Analysis Final Report*, CSC/TM-91/6102, June 1991

Computer Sciences Corporation, *SSE Software Test Management (STM) Capability: Using STM in the Ground Systems Development Environment (GSDE)*, February 1992

Miller, Edward, "Automated Software Testing: a Technical Perspective", *American Programmer*, April 1991

NASA JSC, *DA3 Software Development Metrics Handbook, Version 1*, JSC-25519, December 1991

Section 2 - Requirements for CM

The requirements presented in this section are not intended to constrain the development methods used by the SSCC and SSTF contractors in generating and testing software. The purpose is to specify a set of end-state conditions that are essential to the quality and maintainability of software. The procedures used to achieve those end-state conditions are not specified by this report. Feasible approaches are suggested in part to clarify the requirements, and to illustrate the desired end-state conditions.

2.1 Configuration Management Goals

The requirements specified in this report are intended to achieve two major goals in ground systems software development. This first goal is to minimize obstacles to sustaining engineering of operational software; this is to be achieved by controlling all files that are required to regenerate such software. The second goal is improved overall project productivity; this should result from managing the software interfaces between programmers, between teams, and between releases of software.

2.1.1 Source Management

The first goal fits the traditional view of CM. Record-keeping and change control serve as preventive measures needed because of the extreme ease of modifying software. CM is essential to the long-term management of software as it is developed, maintained, and enhanced.

A depressingly common occurrence in the software industry is the discovery that the operational software can't be maintained (corrected or enhanced) because it doesn't match the source. The thriving market in source code control systems attests to the prevalence of the problem. The problem is particularly common when multiple releases are fielded in parallel; the newest release may be under control, but the source code for previous releases has been "improved" and does not match the executable code. The "improved" source may have additional features, but it may also not integrate with the execution context of the operational software. The alternatives are to back-out the changes, or to forego sustaining engineering on the operational version of the software.

The cost of re-synchronizing source and executable may outweigh any expected benefit from corrections or enhancements, leading to continued use of flawed software. Effective CM can reduce or even eliminate this problem by ensuring that source code exists (or can be regenerated) for all versions of all fielded releases.

Occasionally the source-executable mismatch is due to evolution of related or supporting software (e.g., system services). In the fast-changing world of commercial software (particularly in the areas of open systems and man-machine interfaces), stability is difficult to achieve. Interactions among system elements may make it impossible to roll back to a previous environment. Even so, knowledge of what system elements, such as compilers and libraries, were used in a given build can assist in maintainers in anticipating maintenance problems.

While it may not always be possible to *recreate* a previous environment (especially if more than one upgrade has occurred), it is possible to *characterize* the environment, and thereby manage the potential impact to delivered software.

Most long-lived software exists in several simulatneous versions during its operational life. A particular concern involving multi-version software is that a correction discovered for one version may be unevenly applied to others. Perhaps the source is changed but the executable is not delivered to operations. Perhaps the change is scheduled but not carried out, and then the maintainer is reassigned. Careful CM and archiving of frozen versions, or "snapshots", can alleviate this problem.

2.1.2 Interface Management

The second goal of CM is improvement of overall project productivity. During integration and verification, stability of the test software is extremely important. Unreported changes can cause delays and wasted effort. Time, momentum, and morale can all be squandered tracking down errors traceable to software that was changed without notification. Configuration control may be a burden to individuals, but at the project level it is critically important.

An essential difference between small and large projects is that on the latter, developers can actually degrade project-level performance when they maximize their individual productivity. On small projects the potential impact of unreported changes is always local. Problems can be tracked down without undue difficulty. On large projects the impact may be felt too far away from the cause for easy resolution. The need for "bureaucratic" procedures such as check-out/check-in and independent logging of changes increases with the size of the project.

The costs of CM in interfacing, coordinating, and sharing software are borne at the individual and small-group level, while the benefits may only be obvious at the subsystem or project level. Configuration management is a mechanism for imposing order in what could be a chaotic environment.

The importance of this project-level control is obvious in theory, but rigorous control is sometimes hard to apply in practice. The key role of individual talent in successful projects is well known; the first rule for project success is "Get good people". It seems counter-intuitive to assemble a good team and then to put obstacles in the way of top individual performance. The reporting and approval requirements of CM are often felt to pose such obstacles. Nevertheless without good CM the time saved in unfettered "SLOC-slinging" will be lost many times over in trying to make different pieces fit together, and in tracing down obscure errors during integration. CM is a *project-level* necessity for quality and productivity, and needs to be explained (and sold) to developers in that context.

The requirements presented in the next section are intended to define mechanisms by which these two goals can be achieved in the specific environments of the SSTF and the SSCC.

2.2 Requirements Definition and Analysis

The following requirements are presented in terms of the conditions that should be true as a result of CM activities. The mechanisms by which these conditions are achieved are not prescribed; however, possible approaches are suggested by way of explaining the requirements.

The scope of these requirements is essentially the scope of GS/SPF involvement in ground systems software, that is, from acceptance test through system and integration test to qualification test (QT) and delivery of executable software to operations. The focus of these requirements is on the source code, and the overall goal is the certain knowledge that there is source code on the GS/SPF for all operational software. A secondary goal is knowledge of the status of all anomalies discovered and all changes approved for controlled software.

A basic principle of configuration management and control is that CM responsibility is independent of the software developer (persons or organizations). This independence is similar to the independent role that is generally accepted practice for quality assurance. The requirements below reflect this principle in specifying that NASA-managed CM must be used to satisfy the requirements for delivered software.

2.2.1 CM Specifications

The requirements presented here include both the specifications, in sans-serif type, and explanatory material, in normal (serif) type. The explanations are not part of the formal requirements.

These specifications make use of a concept referred to as a "snapshot", which is an image of a system or element at a given point in time. A snapshot is similar to a frozen subsystem on Rational computers. Snapshots are used in the SSE STM to provide fixed points of reference as software is developed and tested. This set of requirements assumes that snapshots can be made and archived in the CM system.

1. For any software that is delivered to operations: all software elements necessary to recreate the delivered software, with the exception of mission-specific data and COTS software, shall be under the control of NASA-managed CM. CM records must be kept to demonstrate the traceability from controlled files to operational software.

This requirement does not specify that operational software must be generated directly from NASA-managed CM storage. In practice, contractor-managed object files and libraries may be linked with mission-specific data in the target environment, without recompiling every source file. The requirement is that source code, along with all related files such as compile scripts, build scripts, configuration data, required libraries, and so on be up-to-date in the NASA-managed CM storage. Complete records must exist showing the history of any contractor-managed files used in creating operational software.

The exceptions for mission-specific data and COTS software reflect the fact that these elements are provided by other organizations.

A convenient way to satisfy this requirement would be to stage the software and archive a snapshot, using STM. The step of compiling and linking the source files would generally not be required, as long as there is traceability from formal CM to the contractor-managed files.

2. CM records must be kept to verify that the software that is delivered to operations is the same as the software certified in qualification testing, with the exception that mission specific data will be different for operational software.

   Other than mission specific elements, there should be no differences between the software that is certified and the software that is delivered for operations. This requirement says that traceability must be documented between the controlled source files and the test version of executable software.

   The "stage and snapshot" approach noted above is also applicable here.

3. For any software that is certified by qualification testing: all software elements necessary to recreate the tests, with the exception of COTS software, shall be under the control of NASA-managed CM. CM records must be kept to demonstrate the traceability from controlled files to test software.

   This requirement says that the software used in QT (at least, at the end of QT) must be able to be regenerated from controlled storage. Unlike requirement 1, this *includes* the "mission-specific" component of the executable software as well as test cases and test frameworks. The STM archive capability provides a mechanism whereby this could be done.

   QT is a process that occurs over a period of time, and the software under test may get changed (corrected, enhanced) during this process. This requirement only addresses the final product of the QT process, not all of the intermediate products. While formal CM provides the necessary change and configuration control facilities for this process (particularly in conjunction with STM), contractor-managed CM may also be adequate for control of intermediate steps.

4. **The enforcement mechanisms for changes to controlled software, whether automated or procedural, shall involve permanent records and independent approval.**

   A key element of configuration control is that no changes occur to controlled software without authorization and record-keeping. While automated procedures are the preferred method of maintaining records and control, there may be instances where procedural (manual) controls are more cost-effective. The SSTF contractor, for example, has proposed procedural controls to ensure that development files and qualification-tested files are not intermingled on the reconfiguration computer.

   This requirement says that the procedures, automated or manual, must generate records, and must involve approval from someone other than the developer. (The SSTF example mentioned above specifies such independent approval).

5. **The status of all approved changes to controlled software shall be recorded and accessible. The status of an approved change shall include references to all versions of controlled software that have been, or are scheduled to be, changed.**

   The intent of this requirement is to ensure that the status of approved changes, whether applied, in process, or deferred, can be determined for any given snapshot of controlled software. One of the most common forms of uncertainty that CM is intended to prevent is "What changes have been applied to this item of code?". Approved changes include those generated from CRs, DRs, and (if applicable) STRs.

   This requirement also plays a role in support of test planning and sustaining engineering. It is a fact that the more often a component is changed, the more likely it will be changed again. Frequently changed components are good candidates for extra attention during testing, and careful review of the software and its supporting documentation.

6. **The status of all reported discrepancies involving controlled software shall be recorded and accessible.**

   This requirement extends the historical knowledge of a software component by requiring that any faults discovered after the component is placed under formal control be recorded. The intent of this requirement is to assist in assessment of quality, and to identify components that may need review and perhaps redesign.

   One of the best indicators that a component will have to be changed in the future is that it has been changed in the past. The more faults detected in a component, the more likely it is that further faults will be detected in that component. Evidence from many projects, including NASA ground software, indicated that a sizeable fraction of error corrections introduce further errors or fail to correct the initial fault. A record of all faults detected *after* *completion of acceptance test* provides a good indication of the probable stability of the code.

All formal DRs will, of course, be recorded. This requirement adds the condition that informally reported errors also be recorded.

7. For all software that is delivered to operations: the snapshot of the delivered system shall include all relevant ICDs and IDDs.

   The intent of this requirement is to ensure that maintenance on operational software has a contextual baseline as well as a system baseline. The evolution of systems often leads to changes in interfaces. By recording the interface definitions that are in effect when a system is released, the CM system can facilitate maintenance with explicit indications of interface changes.

2.2.2 Discussion of CM requirements

The primary intent of these CM requirements is to simplify the task of sustaining engineering, by ensuring that critical information doesn't get lost in the press of multiple version developments, changing requirements, and tight deadlines. By the time software reaches operations, the simple defects have already been detected and corrected. The faults that occur will generally be sensitive to operating conditions and combinations of software. In trying to correct such errors, it will be essential to have the correct software to modify and test. It will be very valuable to know whether any similar faults were detected during development or test. The ability to run regression testing with the same testbed that was used for certification will simplify verification of error corrections.

The CM requirements will impose additional work on developers, and may cause distress in that all faults are recorded. Nevertheless, these procedures will help to develop and maintain quality software in a timely fashion. The implementation of these requirements should be automated as much as possible, both to minimize the impact on developers and to assure consistency and completeness. Existing tools such as the SSE CM system and the STM can be used to minimize the cost of implementing CM, with the added benefit of smooth integration with the Build Process tools to be provided in SSE OI 7.0.

Section 3 - Requirements for Software Measurement

The report focuses on measuring software in the post-acceptance-test phase. It assumes that contractors for both the SSCC and the SSTF will have software measurement efforts in place, and that these metrics will add to the contractors' understanding of the development process. NASA has a similar interest in seeing what happens to software after it is accepted. These metrics requirements involve questions of software quality and process improvement. Requirements for project management metrics have been addressed in the Software Metrics Guide developed by Mitre for JSC/DA3.

3.1 Goals of Software Measurement

The software measurement requirements detailed in this section are intended to support specific actions to improve the quality of ground system software without unwarranted impact on the developers. As far as possible, the metrics are based on procedures that do not affect the developers at all (e.g., code analysis of software as it is entered into the CM system), or can be made part of regular reporting activities.

The purpose of the software measurement activity is to gain a clear understanding of the deliverable software. Together with project management metrics, this information will enable NASA and contractors to improve the quality and reliability of current systems while improving the forecasting and management of future systems. Software metrics will be used to characterize different elements of the ground systems, and to correlate that understanding with test resource requirements and maintainability.

3.2 Specific Metrics

This presentation assumes that the metrics specified in the DA3 Software Metrics Guide are being reported. These requirements are in addition to that set.

1. Static source code measures that can be generated by automated code analysis software should be collected when source code is checked in to formal CM. Such measures should include size (e.g, number of source lines, number of blank lines, number of non-blank comment lines, number of comments, number of statements, number of executable statements, number of declaration statements), internal complexity (e.g., McCabe cyclomatic complexity), fan-out (the number of procedures invoked by this procedure), and degree of change from previous version.

   All code that's delivered to formal CM should be analyzed on arrival by use of automated code analyzers. These analyzers will provide a wealth of data without impact to the developers. The reporting of this information will depend on the available resources and specific concerns that are system dependent. The recommended reporting is to roll-up these measures as averages and standard deviations at the subsystem level, with more detailed reporting as required to analyze specific problems.

2. Subsystem interface complexity should be measured using automated analytical tools when a subsystem is submitted to formal CM, and with each substantial change to the subsystem. Interface complexity includes the size of the subsystem, the number of components, the depth and breadth of the invocation tree, and the average fan-out of components to other subsystem components.

   In addition to measurements of internal complexity, the system complexity will be measured by determining the interconnectedness of the system. This measure is a counterpart to internal complexity; decreasing one in a system often increases the other. In general, striking a balance between internal complexity and interface complexity minimizes the errors in the system.

   We recommend reporting these measures at the subsystem level, reporting internal and interface complexity for the same set of components.

3. Test coverage should be measured at appropriate points in the development process and reported during integration and qualification testing. Test coverage metrics include unit test metrics (e.g., per cent of source statements tested, per cent of paths tested) and interface test metrics (e.g., report of all calling-called interface pairs with frequency counts).

   Test coverage is a difficult metric to collect, but automated tools can simplify the process. Detailed test coverage analysis is somewhat intrusive, and best performed in the unit test environment where performance is less likely to be a factor. Integration testing provides the opportunity to determine which components call which during simulated operational conditions.

   Test coverage metrics should be reported to the testers in real-time, to help focus efforts on less-well-tested parts of the system. These metrics should be reported during integration and qualification testing at the subsystem level.

4. The amount and form of documentation that exists for software components should be reported at qualification test. This metric includes the fraction of components with current prologues, PDL, build definition scripts, requirements trace, system description, and user/operator information (as appropriate).

   The availability of current documentation is a major factor in quality and maintainability of software. This metric does not attempt to assesss the quality of the documentation, but it does provide a measure of its availability. Some of the documentation (e.g., prologues) is carried along with the source code but often is not kept up-to-date, Other documentation for a component may consist of a paragraph in the system description. For low-level routines, there may be no appropriate user/operator information. The recommended reporting level is the subsystem, summarized from contractor-collected detailed component data.

5. The change history of each component should be recorded, indicating the amount of new, changed, and deleted code associated with each change action. The information should be collected as formally controlled components are checked back in to the CM system.

   For each distinct component, and rolled up to subsystems, this metric reports the number of changes applied and the extent of changes. Changes to software tend to increase the size of the components and disrupt the logical structure created initially. Components that have grown significantly are a result of several changes should be reviewed for possible redesign.

   Reporting should be at the subsystem or system level with histograms showing the fractions of components with different numbers of changes, and the fractions with appropriate relative growth in size.

6. The number of problem reports (if any) associated with each component should be recorded when a component is initially entered into the formal CM system, and after any period of informal testing (that is, without DR actions). The metrics should include the classification (severity) of the problems.

   This metric is designed to capture the number of problems that are reported outside the formal DR system, but after the software has passed its acceptance tests. We recommend that this metric be reported at the subsystem level using histograms of fractions of components with different numbers of formal and informal problems reported. Because the intent of this metric is to provide some indication of probable future errors. multiple reports of single errors should be treated as single reports.

   The maintenance organization may wish to receive a list of components that have experienced high numbers of problems. Development managers may want to review problem-prone components with a view toward redesigning and reimplementing them (thus resetting the error problem counts).

3.3 Relationship to DA3 Metrics Initiative

During the past year, code DA3 at JSC has worked with ground software contractors to develop a set of metrics for software development. The focus of this effort was to identify a common set of measurements that are already being collected, and raise them to a higher level of visibility. One goal of this effort is to establish better communications between contractors and NASA. The recommended metrics primarily (though not entirely) address the period from design through acceptance test.

The metrics proposed in this report are also somewhat limited in scope, as they are intended to provide direct benefit on current projects. These metrics address the characteristics of the software after it has been accepted for integration testing. For the most part, these metrics can be collected from the software itself or from reporting mechanisms that will exist whether or not these metrics are reported (e.g, the CM system).

Like the DA3 metrics set, these metrics are primarily intended to be reported at the subsystem level. We expect that contractors will choose to collect and analyze these metrics in greater detail internally, but the reporting to NASA is primarily at the subsystem level.

The metrics described in this report should provide a good complement to the DA3 metrics set, adding product metrics to the management metrics defined by DA3.

Glossary

Ada	the primary programming language for the Space Station Freedom Project. Ada is a trademark of the US Department of Defense
C	a programming language commonly used with Unix systems and applications programming
CM	configuration management
COTS	commercial off-the-shelf (usually refers to software or hardware)
CR	change request: a formal request to change a requirement
CSC	Computer Sciences Corporation
DR	discrepancy report: a formal report that a system (in this case, usually software) does not meet its requirement specification
formal CM	the SSE-provided CM system residing on the GS/SPF; it manages the software that has been delivered to NASA, a provides a controlled baseline from which deliveries are made to operations
GS/SPF	Ground Systems/Software Production Facility
GSDE	Ground Systems Development Environment
ICD	interface control document
IDD	interface definition document
JSC	Lyndon B. Johnson Space Center, Houston, Texas
NASA	National Aeronautics and Space Administration
OI	operational increment (the added functionality in a new release)
QT	qualification testing--the last testing stage prior to delivery to operational use (in the GSDE life cycle)
RICIS	Research Institute for Computers and Information Systems
snapshot	a complete stored copy of a software component or subsystem at a specified point in its development; subsequent changes to the software can be compared to the snapshot
SPE	software production environment
SSCC	Space Station Control Center
SSE	software support environment (specifically, the SSFP SSE)
SSFP	Space Station Freedom Project
SSTF	Space Station Training Facility
STM	software test management: a capability of the SSE
STR	system (or software) trouble report: a less formal equivalent to a DR
UHCL	University of Houston - Clear Lake