



Estimating and Controlling Software Fault Content More Effectively



Allen P. Nikora

Autonomy and Control Section
Jet Propulsion Laboratory
California Institute of
Technology

Norman F. Schneidewind

Department of Information
Sciences
Naval Postgraduate School
Monterey, CA

John C. Munson

Department of
Computer Science
University of Idaho
Moscow, ID

**NASA Code Q Software Program Center Initiative UPN 323-08;
Kenneth McGill, Research Lead**

***OSMA Software Assurance Symposium
Sept 4-6, 2002***

The work described in this presentation was carried out at the Jet Propulsion Laboratory, California Institute of Technology. This work is sponsored by the National Aeronautics and Space Administration's Office of Safety and Mission Assurance under the NASA Software Program led by the NASA Software IV&V Facility. This activity is managed locally at JPL through the Assurance Technology Program Office (ATPO).



Agenda



- **Overview**
- **Goals**
- **Benefits**
- **Approach**
- **Status**
- **Current Results**
- **References**



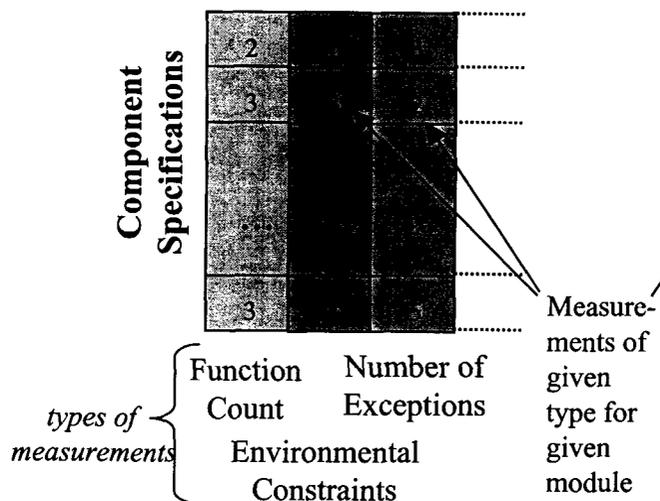
Overview

Objectives: Gain a better quantitative understanding of the effects of requirements changes on fault content of implemented system. Gain a better understanding of the type of faults that are inserted into a software system during its lifetime.

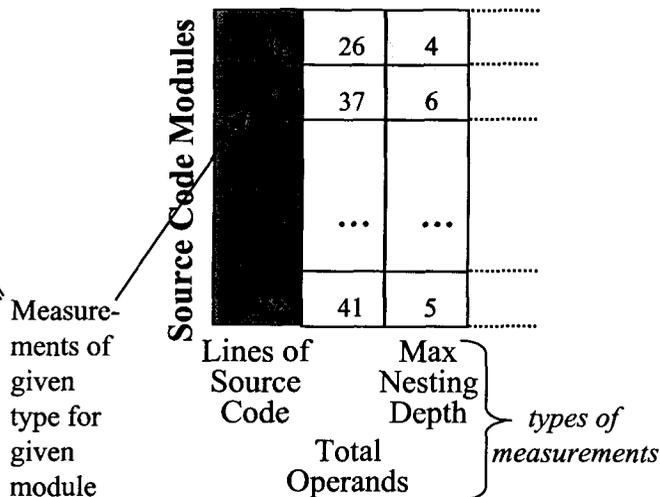
Use measurements to PREDICT faults, and so achieve better

planning (e.g., time to allocate for testing, identify fault prone modules)
guidance (e.g., choose design that will lead to fewer faults)
assessment (e.g., know when close to being done testing)

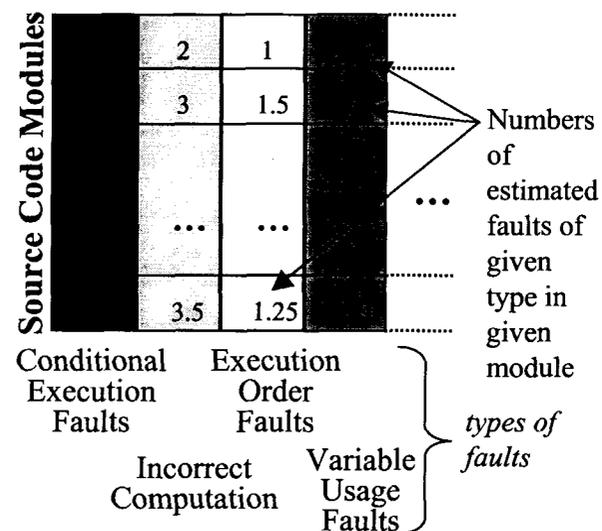
Structural Measurements of Specification



Structural Measurements of Source Code



Estimated Fault Counts by Type for Implemented System





Goals



- **Quantify the effects of requirements changes on the fault content of the implemented system by identifying relationships between measurable characteristics of requirements change requests and the number and type of faults inserted into the system in response to those requests.**
- **Improve understanding of the type of faults that are inserted into a software system during its lifetime by identifying relationships between types of structural change and the number and types of faults inserted.**
- **Improve ability to discriminate between fault-prone modules and those that are not prone to faults.**



Benefits

- **Use easily obtained metrics to identify software components that pose a risk to software and system quality.**
 - **Implementation – identify modules that should have additional review prior to integration with rest of system**
 - **Prior to implementation – estimate impact of changes to requirements on quality of implemented system.**
- **Provide quantitative information as a basis for making decisions about software quality.**
- **Measurement framework can be used to continue learning as products and processes evolve.**



Approach



- **Measure structural evolution on collaborating development efforts**
 - **Initial set of structural evolution measurements collected**
- **Analyze failure data**
 - **Identify faults associated with reported failures**
 - **Classify identified faults according to classification rules**
 - **Identify module version at which each identified fault was inserted**
 - **Associate type of structural change with fault type**



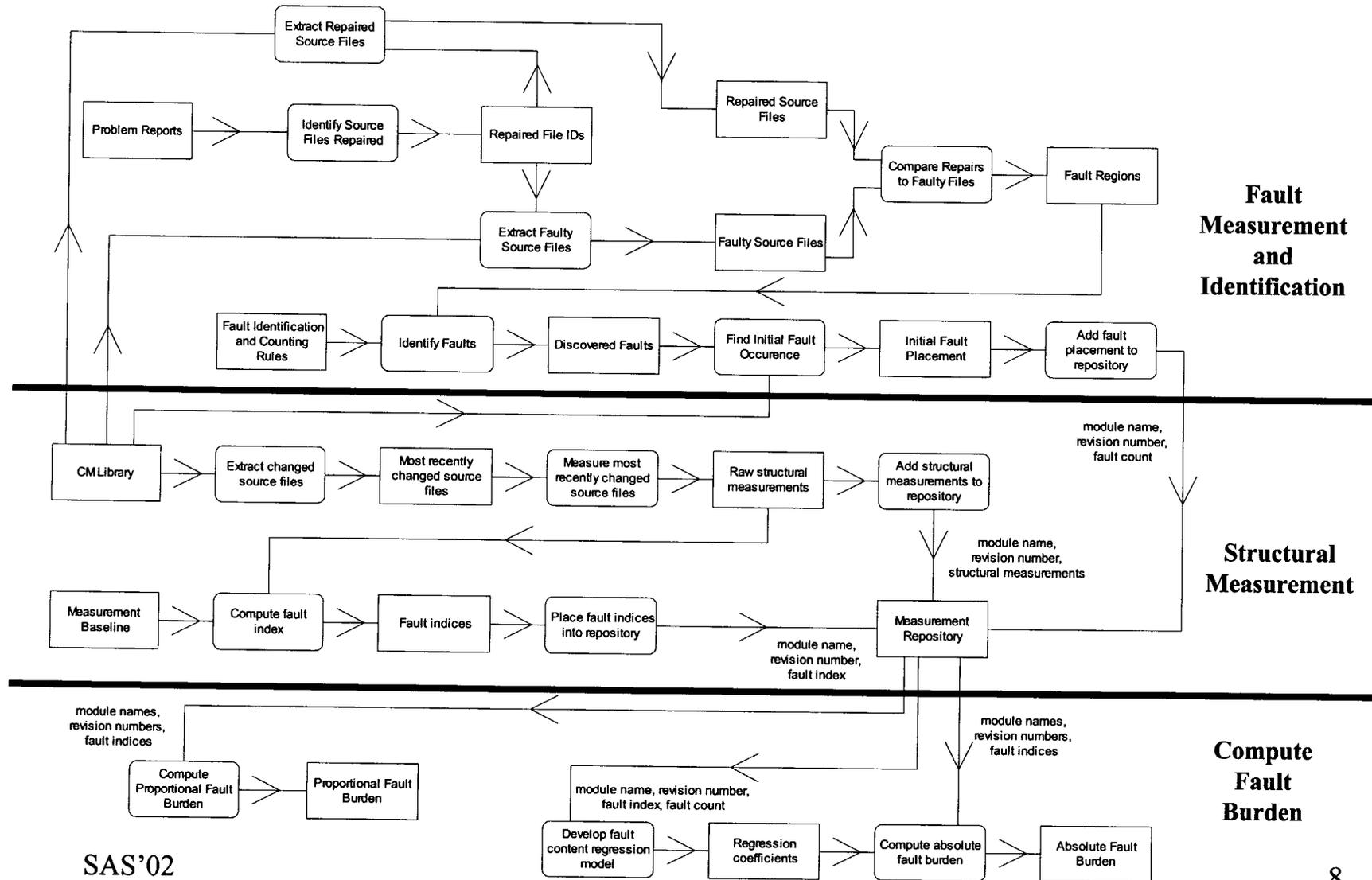
Approach (cont'd)



- **Identify relationships between requirements change requests and implemented quality/reliability**
 - **Measure structural characteristics of requirements change requests (CRs).**
 - **Track CR through implementation and test**
 - **Analyze failure reports to identify faults inserted while implementing a CR**



Approach: *Structural Measurement Framework*





Status



- Year 2 of planned 2-year study
- Investigated relationships between requirements risk and reliability.
- Installed improved version of structural and fault measurement framework on JPL development efforts
 - Participating efforts
 - Mission Data System (MDS)
 - Mars Exploration Rover (MER)
 - Multimission Image Processing Laboratory (MIPL)
 - ***All aspects of measurement framework shown on slide 8 can now be automated***
 - Fault identification and measurement was previously a strictly manual activity
 - Measurement is implemented in DARWIN, a network appliance
 - Minimally intrusive
 - Consistent measurement policies across multiple projects

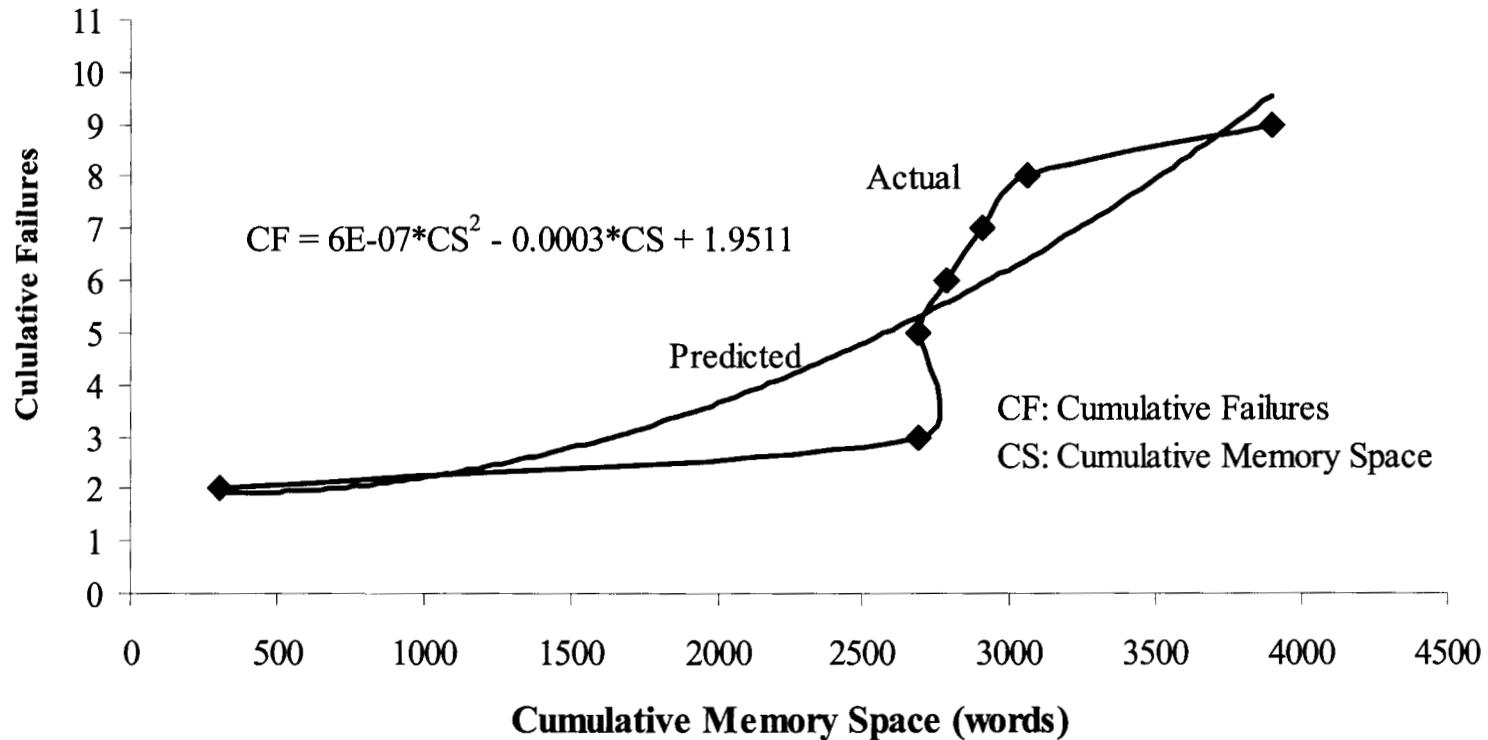


Current Results: ***Requirements Risk vs. Reliability***

- **Analyzed attributes of requirements that could cause software to be unreliable**
 - **Space**
 - **Issues**
- **Identified thresholds of risk factors for predicting when number of failures would become excessive**
- **Further details in [Schn02]**



Current Results: *Requirements Risk vs. Reliability*



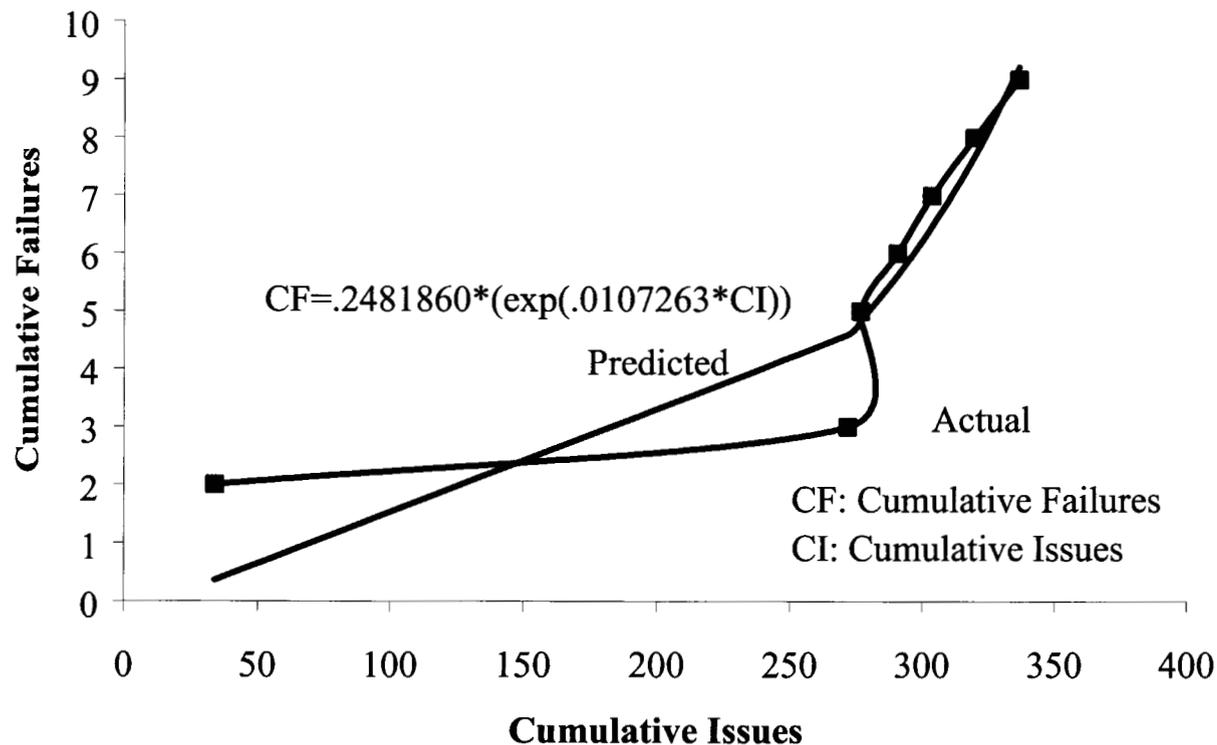
Cumulative Failures vs. Cumulative Memory Space



Current Results:



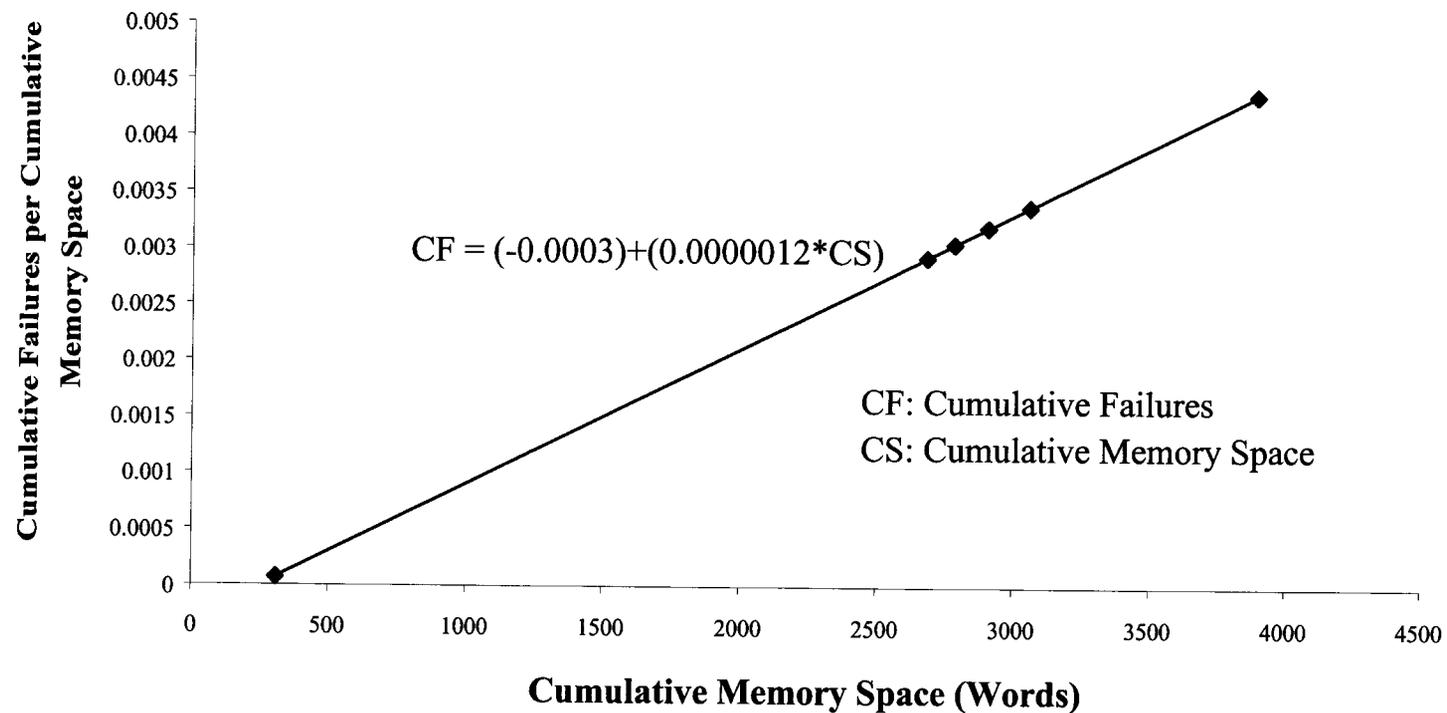
Requirements Risk vs. Reliability



Cumulative Failures vs. Cumulative Issues



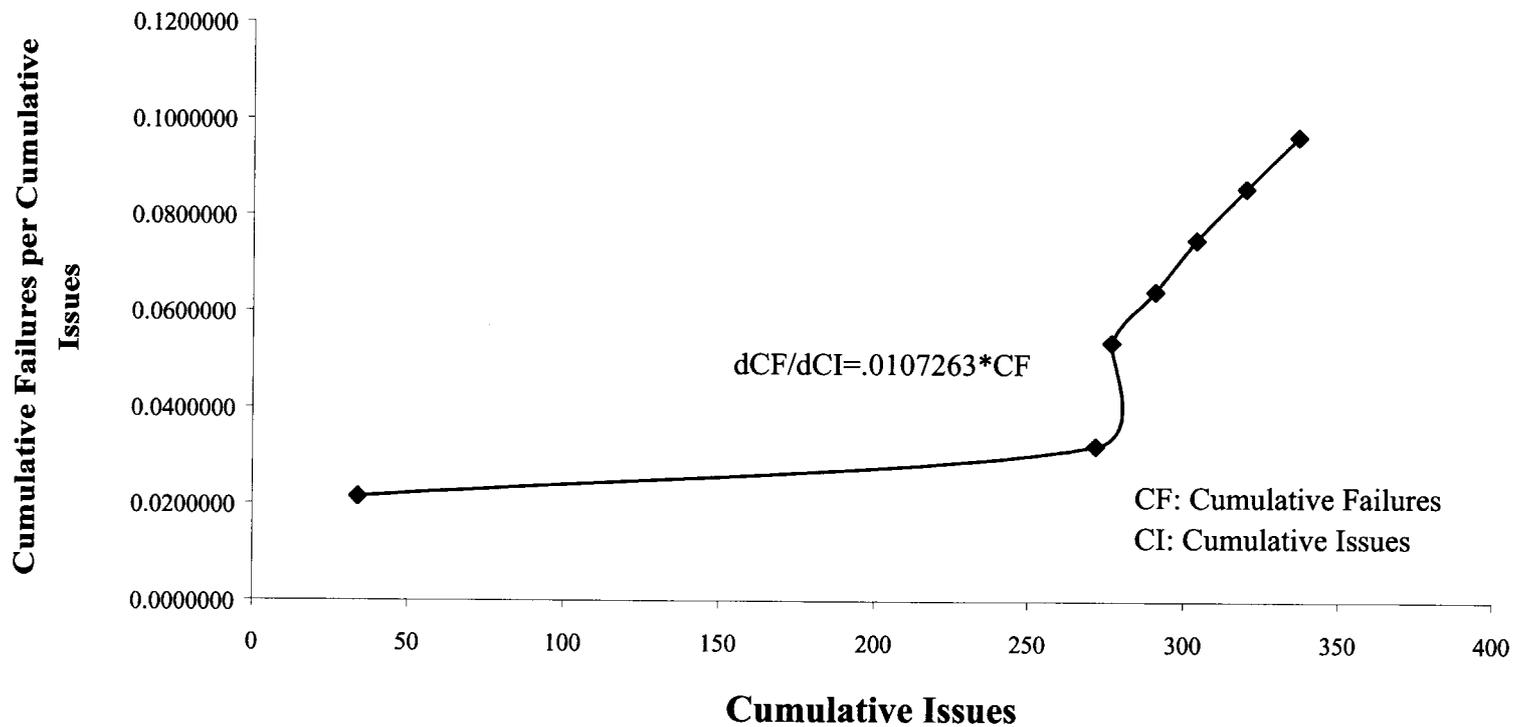
Current Results: *Requirements Risk vs. Reliability*



Rate of Change of Failures with Memory Space



Current Results: *Requirements Risk vs. Reliability*

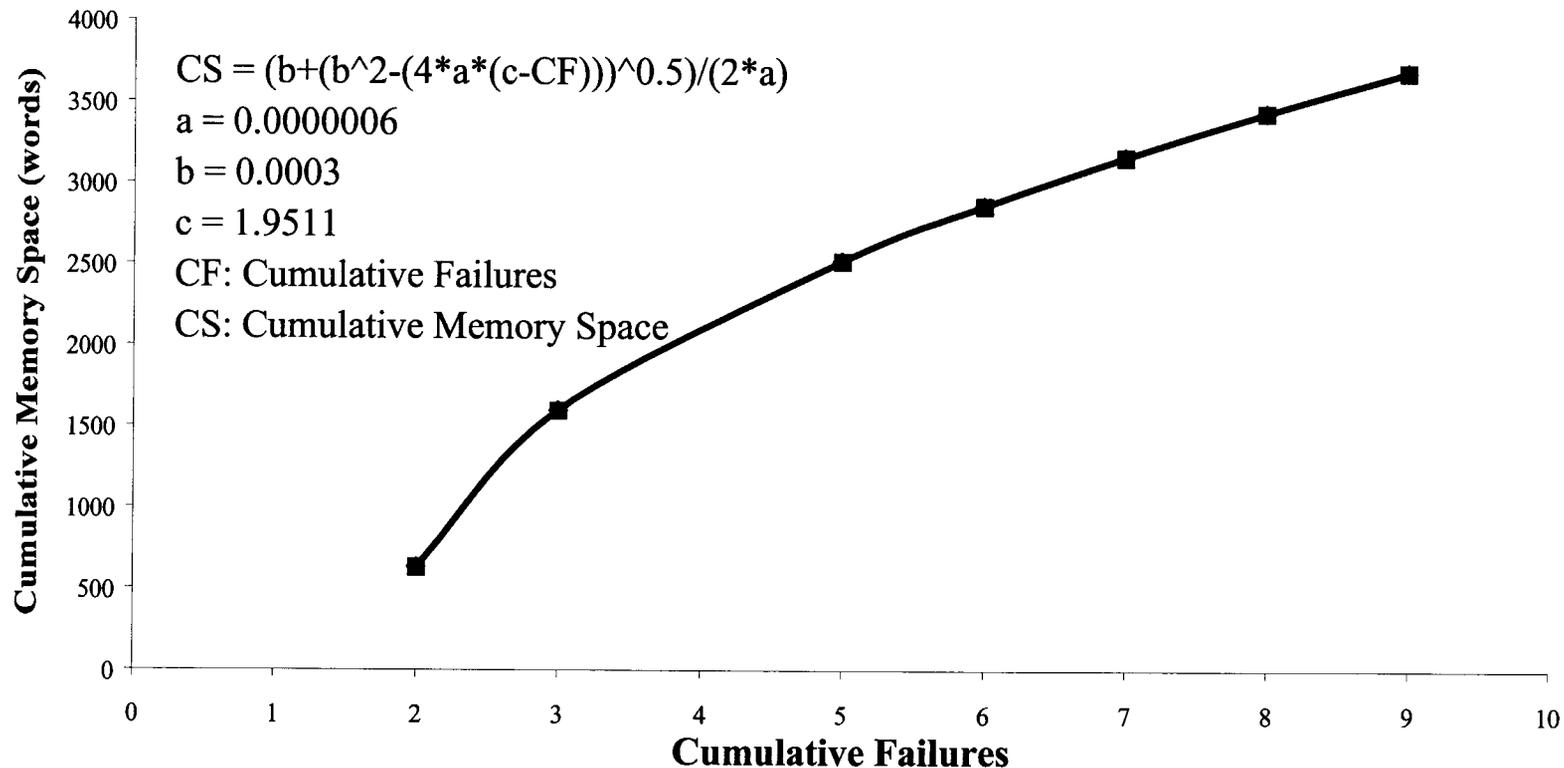


Rate of Change of Failures with Issues



Current Results:

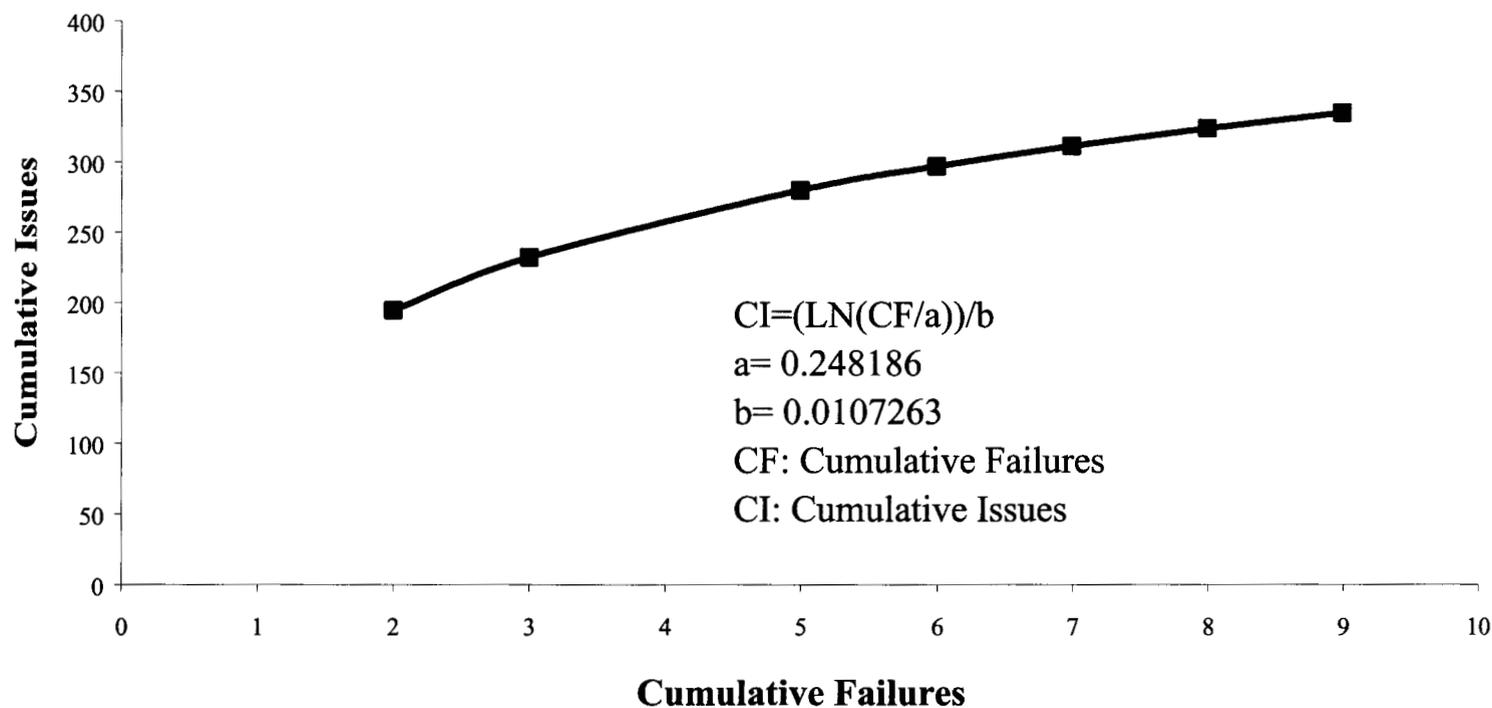
Requirements Risk vs. Reliability



- **Predicting cumulative risk factors**
 - **Cumulative memory space vs. cumulative failures**



Current Results: *Requirements Risk vs. Reliability*



- **Predicting cumulative risk factors**
 - **Cumulative issues vs. cumulative failures**

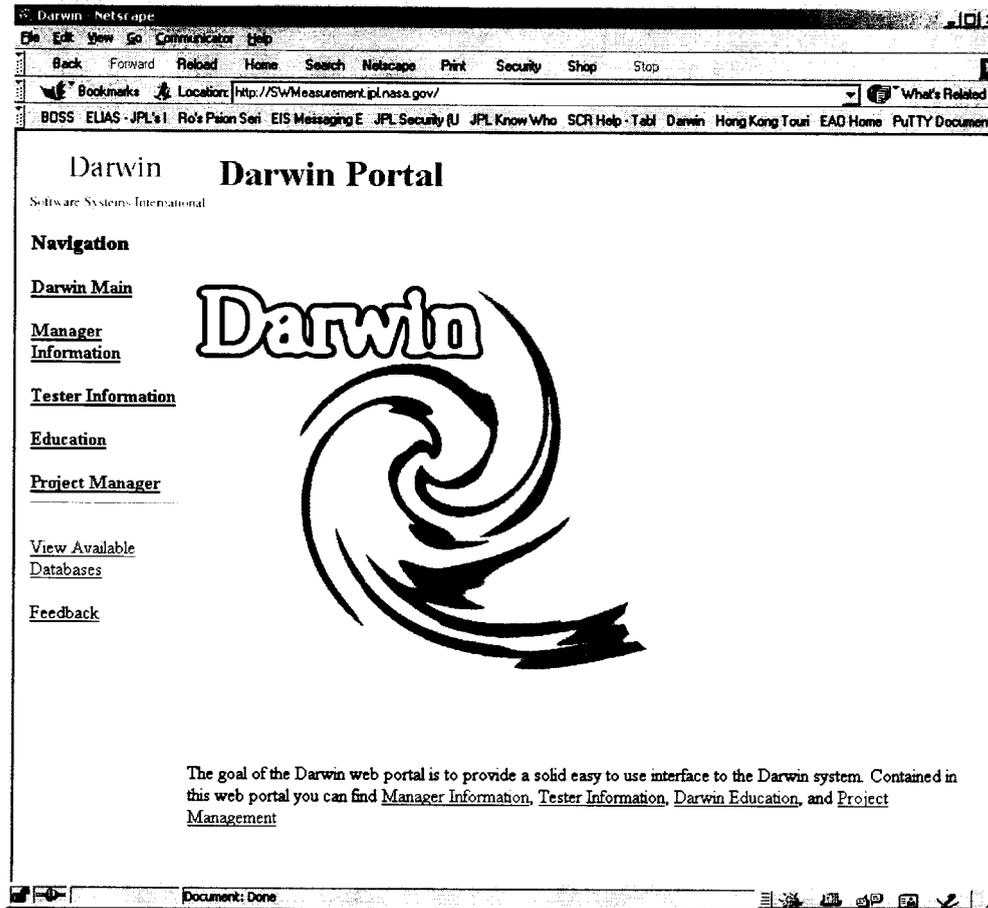


Current Results: *Fault Types vs. Structural Change*

- **Structural measurements collected for release 5 of Mission Data System (MDS)**
 - **1,828 source files**
 - **66,063 unique modules**
 - **2,197,916 total measurements made**
- **Fault index and proportional fault burdens computed**
 - **At system level**
 - **At individual module level**
- **Next slides show typical outputs of DARWIN network appliance**



DARWIN Portal – Main Page



This is the main page of the DARWIN measurement system's user interface.



DARWIN – Structural Evolution Plot

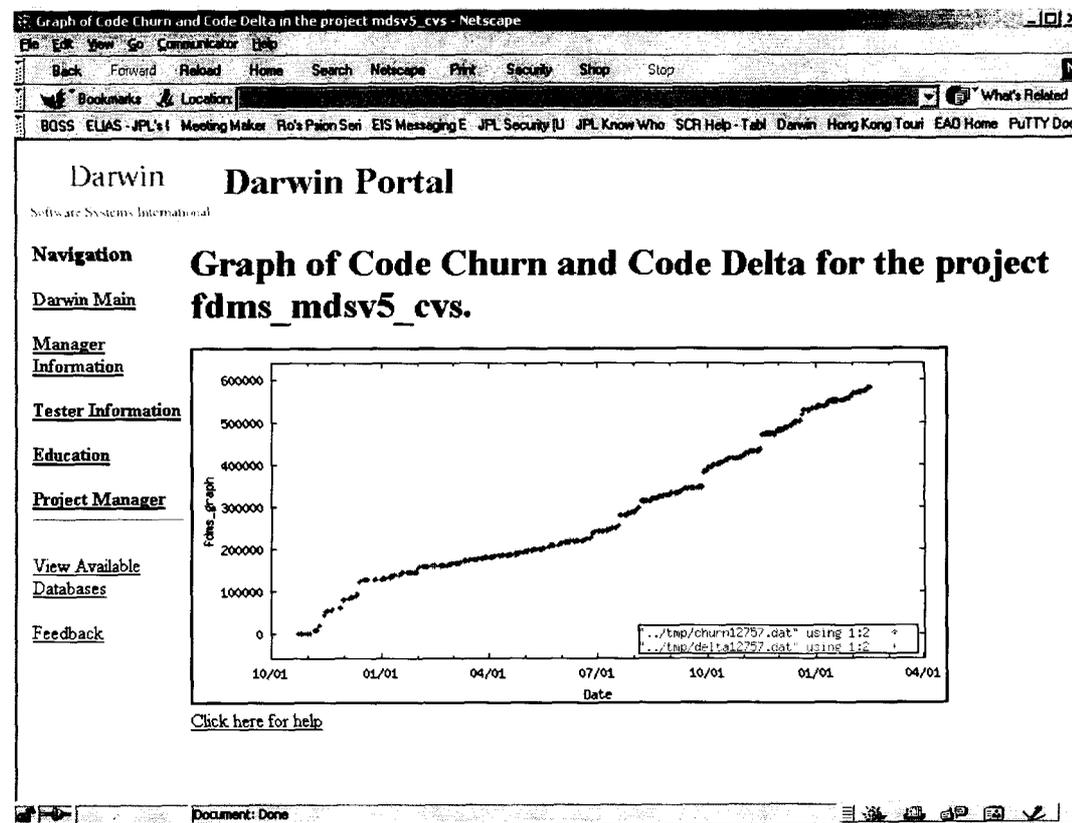


Chart of a system's structural evolution during development. This is available under "Manager Information". Clicking on a data point will bring up a report detailing the amount of change that occurred in each module. This plot shows some of the individual builds for release 5 of the MDS.



DARWIN – Module-Level Build Details



```
(Non-zero) Modules for build 2001-08-09 of project mdsv5_cvs, sorted by Churn since baseline.

-----
test2s()
ParachuteEstimatorTraits::Thread::predictState()
Parameter::getTypeFromStringNoNS(conststd::string&t)
Parameter::getTypeFromString(conststd::string&typ)
TestIntervall::TestIntervall()
mdsmain(constintargc,constchar*argv[])
TestDiscrete::TestDiscrete()
examples()
TestViewFinder::runTests()
SimpleNormalPositionEstimatorTraits::Thread::processMeasurement()
PositionEstimateFunctionTest(Dispatch&r,conststd::string&key,constCGIArgs&args)
AirDragModelParameterEstimatorTraits::Thread::predictState()
Server::getNextRequest()
FifoReaderHandler::start(Dispatch&r,conststring&key,constCGIArgs&args)
GreaseFilterTest(Dispatch&r,conststd::string&key,constCGIArgs&args)
OSTimeService::svc(void)
AttributeSetter::setIt(conststd::string&valueString)
LengthTest(Dispatch&r,conststd::string&key,constCGIArgs&args)
CarExampleMain(intargc,char*argv[],CppUnit::TestSuite&args)
-----
```

	Churn From Baseline
	3108.478289
	2826.317554
	2726.351943
	2725.674228
	2254.132621
	1302.320181
	670.321970
	660.459373
	509.278008
	366.179343
	362.248144
	297.209347
	276.794827
	253.716821
	245.984883
	243.421666
	241.489394
	232.521508
	226.600649

This report shows the amount of change that's occurred for each module shown in this particular build (2001-03-10).



Current Results: *Fault Identification and Measurement*



- **Developing software fault models depends on definition of what constitutes a fault**
- **Desired characteristics of measurements, measurement process**
 - **Repeatable, accurate count of faults**
 - **Measure at same level at which structural measurements are taken**
 - **Measure at module level (e.g., function, method)**
 - **Easily automated**
- **More detail in [Mun02]**



Current Results: *Fault Identification and Measurement*



- **Approach**
 - **Examine changes made in response to reported failures**
 - **Base recognition/enumeration of software faults on the grammar of the software system's language**
 - **Fault measurement granularity in terms of tokens that have changed**



Current Results: *Fault Identification and Measurement*

- **Approach (cont'd)**
 - **Consider each line of text in each version of the program as a bag of tokens**
 - **If a change spans multiple lines of code, all lines for the change are included in the same bag**
 - **Number of faults based on bag differences between**
 - **Version of program exhibiting failures**
 - **Version of program modified in response to failures**
 - **Use version control system to distinguish between**
 - **Changes due to repair and**
 - **Changes due to functionality enhancements and other non-repair changes**



Current Results: *Fault Identification and Measurement*

- **Example 1**
 - **Original statement: $a = b + c$;**
 - $B_1 = \{ \langle a \rangle, \langle = \rangle, \langle b \rangle, \langle + \rangle, \langle c \rangle \}$
 - **Modified statement: $a = b - c$;**
 - $B_2 = \{ \langle a \rangle, \langle = \rangle, \langle b \rangle, \langle - \rangle, \langle c \rangle \}$
 - $B_1 - B_2 = \{ \langle + \rangle, \langle - \rangle \}$
 - $|B_1| = |B_2|, |B_1 - B_2| = 2$
 - **One token has changed \Rightarrow 1 fault**



Current Results: *Fault Identification and Measurement*

- **Example 2**
 - **Original statement: $a = b - c$;**
 - $B_2 = \{ \langle a \rangle, \langle \Rightarrow \rangle, \langle b \rangle, \langle - \rangle, \langle c \rangle \}$
 - **Modified statement: $a = c - b$;**
 - $B_3 = \{ \langle a \rangle, \langle \Rightarrow \rangle, \langle c \rangle, \langle - \rangle, \langle b \rangle \}$
 - $B_2 - B_3 = \{ \}$
 - $|B_2| = |B_3|$, $|B_2 - B_3| = 0$
 - **1 fault representing incorrect sequencing**



Current Results: *Fault Identification and Measurement*

- **Example 3**
 - **Original statement: $a = b - c$;**
 - $B_3 = \{ \langle a \rangle, \langle = \rangle, \langle c \rangle, \langle - \rangle, \langle b \rangle \}$
 - **Modified statement: $a = 1 + c - b$;**
 - $B_4 = \{ \langle a \rangle, \langle = \rangle, \langle 1 \rangle, \langle + \rangle, \langle c \rangle, \langle - \rangle, \langle b \rangle \}$
 - $B_3 - B_4 = \{ \langle 1 \rangle, \langle + \rangle \}$
 - $|B_3| = 6$, $|B_4| = 8$, $|B_4| - |B_3| = 2$
 - **2 new tokens representing 2 faults**



Current Results: *Fault Identification and Measurement*



- **Available Failure/Fault Information**
 - **For each failure observed during MDS testing, the following information is available**
 - **The names of the source file(s) involved in repairs**
 - **The version number(s) of the source files in repairs**
 - **Example on next slide**



Current Results: *Fault Identification and Measurement*



Available Failure/Fault Information – Example

Directory	File name	Version	Problem Report ID
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	CurrentTime.cpp	1	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	make.cfg	4	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	make.cfg	3	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	make.cfg	2	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	RTDuration.cpp	2	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	RTDuration.h	2	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	RTEpoch.cpp	2	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	RTEpoch.h	2	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	testRTDuration.cpp	0	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	TestRTDuration.cpp	1	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	TestRTDuration.cpp	0	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	TestRTDuration.h	2	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	TestRTDuration.h	1	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	TestRTDuration.h	0	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	testRTEpoch.cpp	1	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	TmgtException.cpp	0	IAR-00182
MDS_Rep/source/Mds/Fw/Time/Tmgt/c++/	TmgtException.h	0	IAR-00182



Current Results: *Fault Identification and Measurement*



Fault Identification and Counting Tool Output

MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ArchetypeConnectorFactory.cpp 1 42
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ArchitectureElementDefinition.cpp 1 35
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ArchitectureInstanceRegistry.cpp 1 79
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ArchitectureInstanceRegistry.cpp 2 8
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ArchitectureInstanceRegistry.cpp 3 0
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ArchManagedInstance.cpp 1 36
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.CallableInterface.cpp 1 48
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.CallableInterface.cpp 2 3
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.CGIMethodRegistration.cpp 1 4
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.Collection.cpp 1 12
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.Collection.cpp 2 37
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ComponentComponentLinkInstance.cpp 1 0
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ComponentComponentLinkInstance.cpp 2 65
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ComponentConnectorLinkInstance.cpp 1 0
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ComponentConnectorLinkInstance.cpp 2 50
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ComponentObjectLinkInstance.cpp 1 27
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ComponentObjectLinkInstanceArguments.cpp 1 0
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ComponentRegistration.cpp 1 2
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ConcreteComponentInstance.cpp 1 8
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ConcreteComponentInstance.cpp 2 0
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ConcreteConnectorInstance.cpp 1 42
MDS Fault count/MDS Rep.source.Mds.Fw.Car.c++.ConcreteConnectorInstance.cpp 2 27

Output format:

<Source file name> <source file version> <fault count>



References and Further Reading



- [Mun02]** J. Munson, A. Nikora, “Toward A Quantifiable Definition of Software Faults”, to be published in the proceedings of the International Symposium on Software Reliability Engineering, Annapolis, MD, November 12-15, 2002
- [Schn02]** N. Schneidewind, “Requirements Risk versus. Reliability”, to be presented at the International Symposium on Software Reliability Engineering, Annapolis, MD, November 12-15, 2002
- [Nik02]** A. Nikora, M. Feather, H. Kwong-Fu, J. Hihn, R. Lutz, C. Mikulski, J. Munson, J. Powell, “Software Metrics In Use at JPL Applications and Research”, 8th IEEE International Software Metrics Symposium, June 4-7, 2002, Ottawa, Ontario, Canada
- [Nik02a]** A. Nikora, J. Munson, “Automated Software Fault Measurement”, Assurance Technology Conference, Glenn Research Center, May 29-30, 2002
- [Schn01]** Norman F. Schneidewind, “Investigation of Logistic Regression as a Discriminant of Software Quality”, proceedings of the International Metrics Symposium, 2001
- [Nik01]** A. Nikora, J. Munson, “A Practical Software Fault Measurement and Estimation Framework”, Industrial Practices presentation, International Symposium on Software Reliability Engineering, Hong Kong, November 27-30, 2001



References and Further Reading (cont'd)



- [Schn99] N. Schneidewind, A. Nikora, "Predicting Deviations in Software Quality by Using Relative Critical Value Deviation Metrics", proceedings of the 10th International Symposium on Software Reliability Engineering, Boca Raton, FL, Nov 1-4, 1999
- [Nik98] A. Nikora, J. Munson, "Software Evolution and the Fault Process", proceedings, 23rd Annual Software Engineering Workshop, NASA/Goddard Space Flight Center, Greenbelt, MD, December 2-3, 1998
- [Schn97] Norman F. Schneidewind, "A Software Metrics Model for Quality Control", Proceedings of the International Metrics Symposium, Albuquerque, New Mexico, November 7, 1997, pp. 127-136.
- [Schn97a] Norman F. Schneidewind, "A Software Metrics Model for Integrating Quality Control and Prediction", Proceedings of the International Symposium on Software Reliability Engineering, Albuquerque, New Mexico, November 4, 1997, pp. 402-415.