Infusing Software Fault Measurement and Modeling Techniques

Allen Nikora
Software Quality Assurance Group
Jet Propulsion Laboratory
California Institute of Technology

Al Gallo
Software Assurance Technology Center
Goddard Space Flight Center
Greenbelt, MD

John 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 2003

July 30 – Aug 1, 2003

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
- Papers and Presentations Resulting From This CI

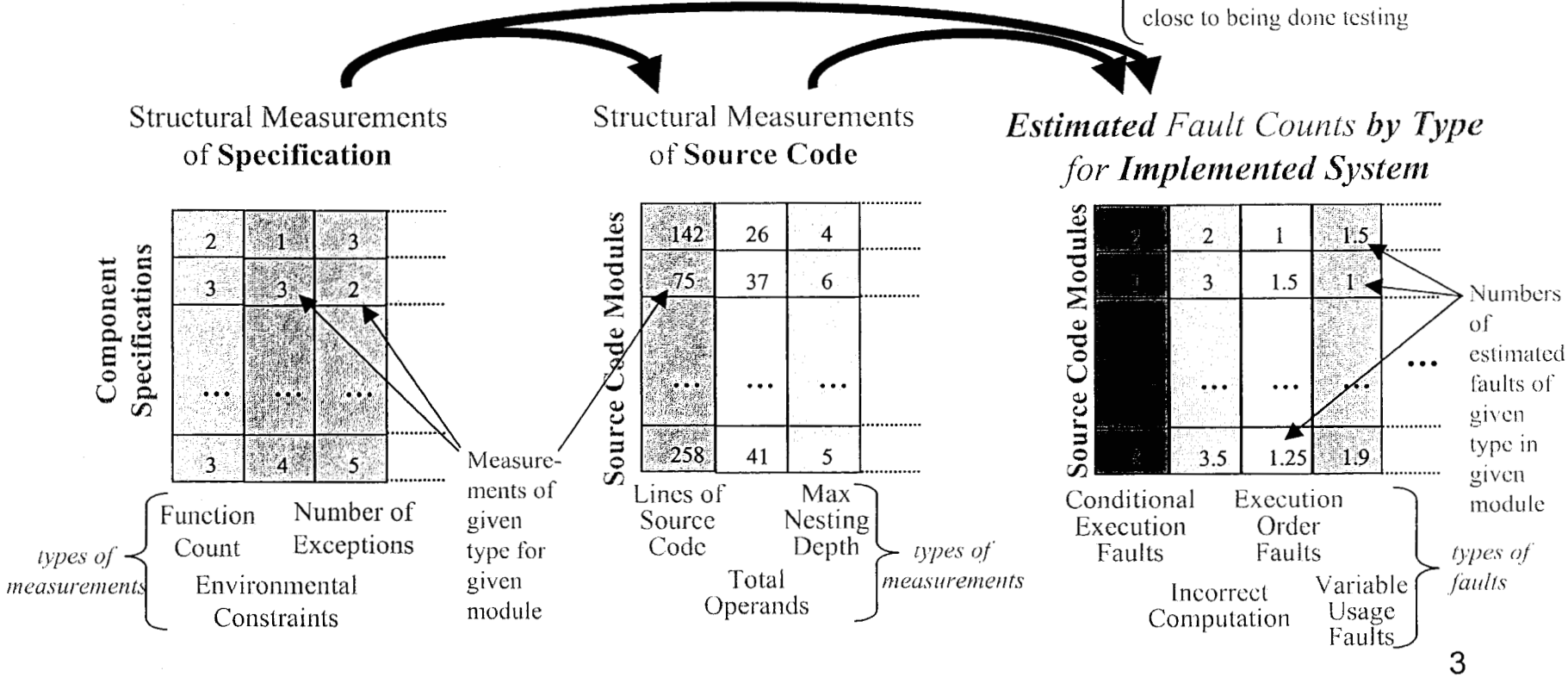
Overview

California Institute of Technology

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



California Institute of Technology

Goals

- Develop a viable method of infusing the measurement and fault modeling techniques developed during the first two years of this task into software development environments at GSFC and JPL
  - Collaborate with SATC and selected projects at GSFC
  - Continue and extend collaboration with projects at JPL
- Develop training materials for software measurement for software engineers/software assurance personnel
  - Measurement background
  - Using DARWIN Network Appliance
  - Organizational interfaces
  - Interpreting output

Benefits

- Provide quantitative information as a basis for making decisions about software quality.
- Use easily obtained metrics to identify software components that pose a risk to software and system quality.
- Measurement framework can be used to continue learning as products and processes evolve.

Approach

- Measure structural evolution on collaborating development efforts
  - Structural measurements for several JPL projects collected
  - Several GSFC projects have shown interest
- Analyze failure data
  - Identify faults associated with reported failures
    - Relies on:
      - All failures being recorded
      - Failure reports specifying which versions of which files implement changes responding to the reported failure.
  - Count number of repaired faults according to token-count technique reported in ISSRE'02 [Mun02]. (Fault count is dependent variable)
- Analyze relationships between number of faults repaired and measured structural evolution during development

Approach (cont'd)

California Institute of Technology

- 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
- Develop training materials for software measurement for software engineers/software assurance personnel
  - DARWIN user's guide nearly complete
  - Measurement class materials being prepared

Status

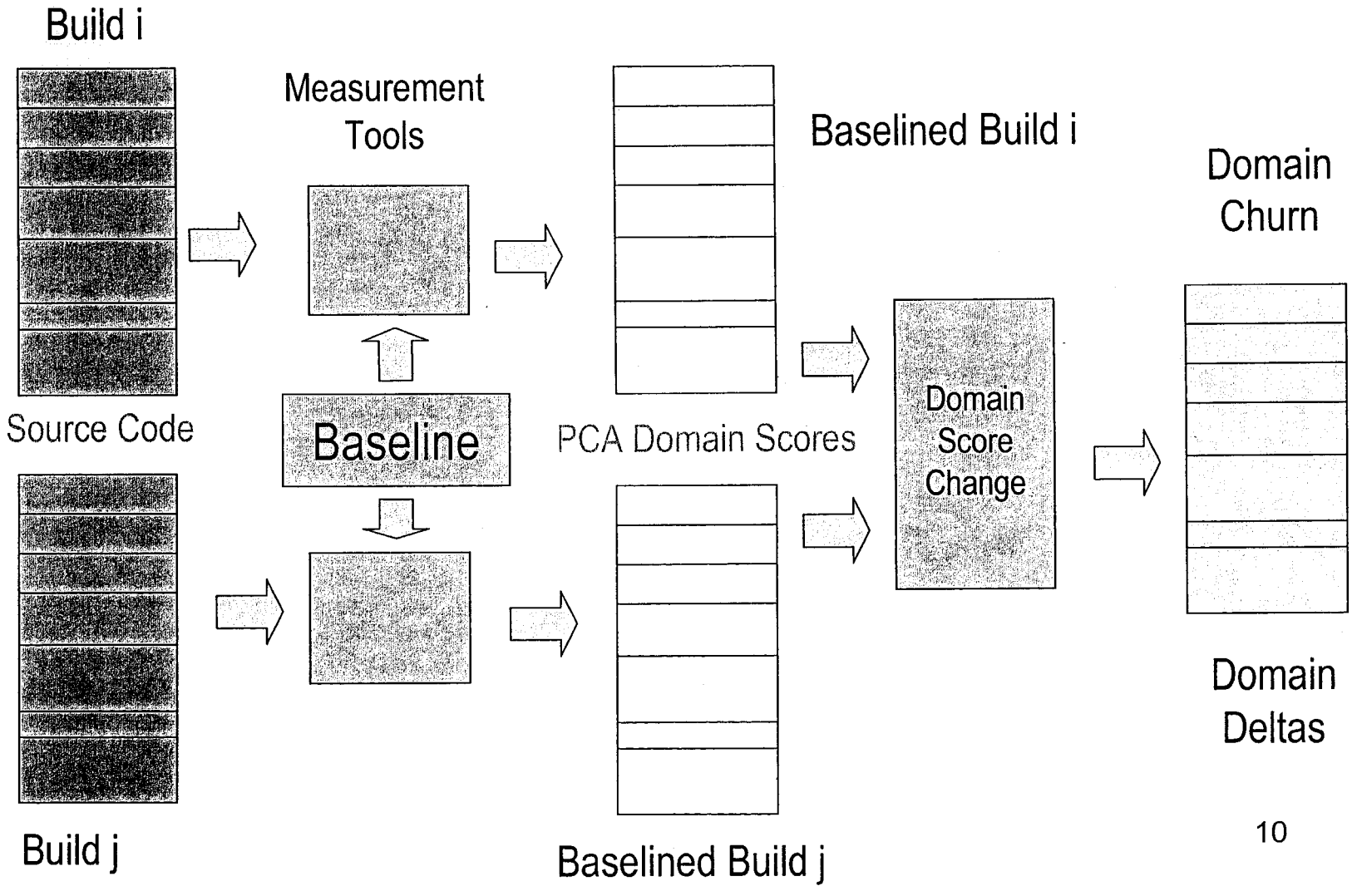
- Follow-on to previous 2-year effort, "Estimating and Controlling Software Fault Content More Effectively".
- Investigated relationships between requirements risk and reliability.
- Installed improved version of structural and fault measurement framework on JPL development efforts
  - Participating efforts
    - Mission Data System (MDS)
    - Mars Exploration Rover (MER)
    - Multimission Image Processing Laboratory (MIPL)
    - GSFC efforts
  - *All aspects of measurement are now 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: *Measuring Software Structural Evolution*

JPL
California Institute of Technology

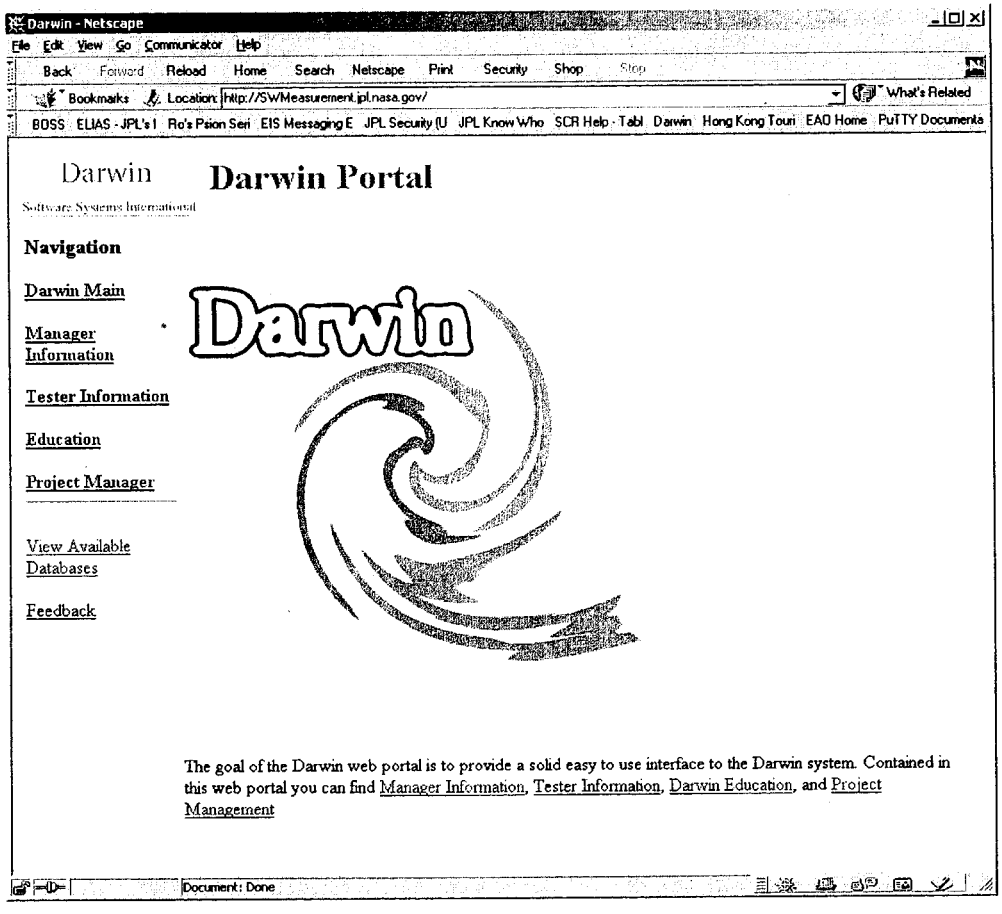
- Mars Exploration Rover (MER)
- Multimission Image Processing Laboratory (MIPL)
- Mission Data System (MDS)
  - Structural measurements collected for release 5 of MDS
    - > 1500 builds
    - > 65,000 unique modules
  - Domain scores, "domain churn", and proportional fault burdens computed
    - At system level
    - At individual module level
  - > 1,400 anomaly reports analyzed

Current Results: *Measuring Software Structural Evolution*



DARWIN Portal – Main Page

California Institute of Technology



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

DARWIN – Structural Evolution Plot

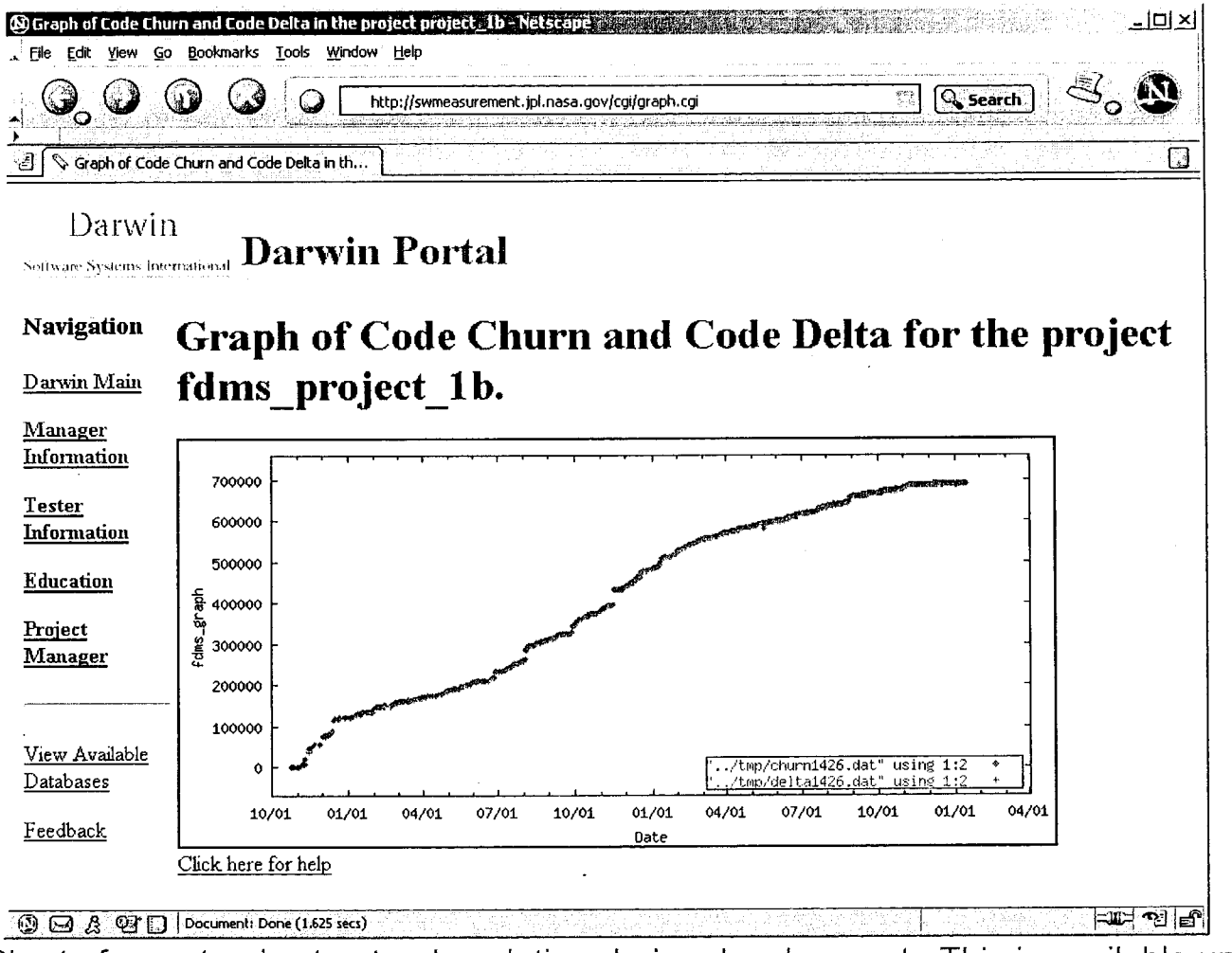


Chart of a system's structural evolution during development. This is available under "Manager Information". Clicking on a data point will bring up a report detailing the amount of change that occurred in each module. This plot shows some of the individual builds for release 5 of the MDS.

DARWIN – Module-Level Build Details

California Institute of Technology

(Non-zero) Modules for build 2002-02-07 of project project_1b, sorted by Churn since baseline.

Modulename	Churn From Baseline
nearpoint(ostream& error)	789.413504
stumpff03(double x, double& c0, double& c1, double& c2, double& c3)	525.230847
coarseNear(const Mds::Fw::Math::Dbl3Vec& pos, Mds::Fw::Math::Dbl3Vec& near, double& lambda)	515.488980
factors(Axis a0, Axis a1, Axis a2, Quaternion& q0, Quaternion& q1, Quaternion& q2)	459.185853
validInputs(ostream& error)	349.297666
doContent(XML_Parser parser, int startTagLevel, const ENCODING *enc, const char *s, const char *end, const char **nextPtr)	329.523165
cartes2sphere(Dim dim, const Mds::Fw::Math::Dbl3Vec& cartesPos, const Mds::Fw::Math::Dbl3Vec& cartesVel, const Mds::Fw::Math::Dbl3Vec& cartesAcc, double* v)	319.309469
constructors(ostream& error)	318.905140
doProlog(XML_Parser parser, const ENCODING *enc, const char *s, const char *end, int tok, const char *next, const char **nextPtr)	311.585820
examples()	309.099387
getSixDOF(const Mds::Fw::Time::Tmgt::RTEpoch& time)	301.170203
check(ostream& error, const char* test, const Position& p, const Mds::Fw::Math::Dbl3Vec& rightPos)	300.133324
dAngleAndAxis(double& angle, double& dAngle, double& d2Angle, Mds::Fw::Math::Dbl3Vec& axis, Mds::Fw::Math::Dbl3Vec& dAxis, Mds::Fw::Math::Dbl3Vec& d2Axis)	296.781369
processMeasurementAndPredict(const Mds::Fw::Time::Tmgt::RTEpoch& current, const Mds::Fw::Time::Tmgt::RTEpoch& stop)	289.353008
test2s()	279.141671
addition(ostream& error)	274.478272
multiplication(ostream& error)	270.533463
ellipsoid(ostream& error)	266.891307
TestDiscrete::TestDiscrete()	261.403950
TestIntervallic::TestIntervallic()	252.444604
polishedNear(const Mds::Fw::Math::Dbl3Vec& pos, Mds::Fw::Math::Dbl3Vec& near, double& lambda)	247.985278
subtraction(ostream& error)	244.748985
storeAtts(XML_Parser parser, const ENCODING *enc, const char *attStr, TAG_NAME *tagNamePtr, BINDING **bindingsPtr)	240.951958
GreaseFilterTest(Dispatch& r, const std::string& key, const CGIArgs& args)	240.699311
quaternions(ostream& error)	239.337829
doTest()	239.029389
cartes2cylin(Dim dim, const Mds::Fw::Math::Dbl3Vec& cartesPos, const Mds::Fw::Math::Dbl3Vec& cartesVel, const Mds::Fw::Math::Dbl3Vec& cartesAcc, double* v)	229.384465
carmain(int argc, char* argv[])	225.550063

This report shows the amount of change that's occurred for each module shown in this particular build (2002-02-07).

Current Results: *Measuring Software Structural Evolution*

JPL
California Institute of Technology

Metric	Definition
Exec	Number of executable statements
NonExec	Number of non-executable statements
N_1	Total operator count
η_1	Unique operator count
N_2	Total operand count
η_2	Unique operand count
Nodes	Number of nodes in the module control flow graph
Edges	Number of edges in the module control flow graph
Paths	Number of paths in the module control flow graph
MaxPath	The length of the path with the maximum edges
AvePath	The average length of the paths in the module control flow graph
Cycles	Total number of cycles in the module control flow graph

Standardized definitions were developed for each measurement

Current Results: *Measuring Software Structural Evolution*

JPL
California Institute of Technology

Metric	Domain		
	1	2	3
Exec	.60	.49	.47
NonExec	.64	.53	.18
N_1	.28	.64	.65
η_1	.49	**.70**	.07
N_2	.28	.64	.65
η_2	.35	**.90**	.04
Nodes	**.87**	.31	.27
Edges	**.88**	.31	.27
Paths	.17	-.10	**.89**
MaxPath	**.87**	.35	.29
AvePath	**.86**	.34	.33
Cycles	**.67**	.22	-.02
Eigenvalues	4.79	3.13	2.24

Table above shows measurement domains resulting from PCA

Current Results: *Fault Identification and Measurement*

JPL
California Institute of Technology

- 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*

JPL
California Institute of Technology

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*

JPL
California Institute of Technology

- 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*

JPL California Institute of Technology

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

Current Results: *Fault Identification and Measurement*

JPL California Institute of Technology

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

Current Results: *Fault Identification and Measurement*

JPL California Institute of Technology

Example 3

- Original statement: a = b - c;
  - B_3 = {<a>, <=>, <c>, <->, <b>}
- Modified statement: a = 1 + c - b;
  - B_4 = {<a>, <=>, <1>, <+>, <c>, <->, <b>}
- $B_3 - B_4$ = {<1>, <+>}
- $|B_3| = 6$, $|B_4| = 8$, $|B_4| - |B_3| = 2$
- 2 new tokens representing 2 faults

Current Results: *Fault Identification and Measurement*

JPL
California Institute of Technology

- 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*

California Institute of Technology

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*

California Institute of Technology

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>

Current Results: *Modeling Fault Content*

JPL
California Institute of Technology

- Fault models developed from:
  - Measured structural evolution (cumulative amount of change for each module).
  - Number of faults repaired for each module.
- Analysis indicates that the amount of structural evolution is related to the number of faults repaired [Nik03].

Current Results: *Modeling Fault Content*

JPL California Institute of Technology

Regression ANOVA

Source	Sum of Squares	df	Mean Square	F	Sig.
Regression	10091546	3	3363848	293	P<.01
Residual	6430656	560	11483		
Total	16522203	563			

Regression Model

Model	Coefficients	t	Sig.
(Constant)	18.24	3.5	P<.01
Domain 1 Churn	21.63	17.3	P<.01
Domain 2 Churn	-.59	-.3	p>.01
Domain 3 Churn	.93	.7	p>.01

Quality of the Regression Model

	R	R Square	Adjusted R Square	Std. Error of the Estimate
Model Summary	.782	.611	.609	107.16024

Current Results: *Modeling Fault Content*

California Institute of Technology

Fault Counting Method vs. Model Quality

- Which fault counting methods produce better fault models?
  - Number of tokens changed
  - Number of "sed" commands required to make each change
  - Number of modules changed

Current Results: *Modeling Fault Content*

JPL California Institute of Technology

Fault Counting Method vs. Model Quality

Number of tokens changed

Model	Coefficients	t	Sig.
(Constant)	18.24	3.5	p<.05
Domain 1 Churn	21.63	17.3	p<.05
Domain 2 Churn	-.59	-.3	p>.05
Domain 3 Churn	.93	.7	p>.05

Number of "sed" commands

Model	Coefficients	t	Sig.
(Constant)	2.484	14.555	p<.05
Domain 1 Churn	.151	3.411	p<.05
Domain 2 Churn	.529	6.489	P<.05
Domain 3 Churn	-0.087	-1.791	p>.05

Number of modules changed

Model	Coefficients	t	Sig.
(Constant)	1.200	35.995	p<.05
Domain 1 Churn	0.009	1.041	p>.05
Domain 2 Churn	0.143	8.920	p<.05
Domain 3 Churn	-0.043	-4.483	p<.05

Current Results: *Modeling Fault Content*

JPL California Institute of Technology

Fault Counting Method vs. Model Quality

Model	R Square	Adjusted R Square	Std. Error of the Estimate
Model 1 – tokens changed	0.61	.061	107.16
Model 2 – number of "sed" commands	0.19	0.19	3.88
Model 3 – number of changed modules	0.14	0.14	0.76

Comparison of Model Quality

Current Results: *User's Guide*

Table of Contents

1. Introduction
    - 1.1. Purpose
    - 1.2. Scope
2. Measuring a Software System's Structural Evolution
    - 2.1. Data Requirements
    - 2.2. Creating a New Project in Darwin
    - 2.3. Importing the Source Repository
    - 2.4. Initial Structural Measurement
    - 2.5. Analyzing Structural Evolution
        - 2.5.1. Establishing a Baseline
        - 2.5.2. Computing Structural Change
    - 2.6. Structural Measurement Outputs
3. Measuring the Number of Faults Repaired
    - 3.1. Data Requirements
    - 3.2. Identifying Changes
    - 3.3. Counting Faults
    - 3.4. Fault Measurement Outputs
    - 3.5. Potential Sources of Noise
4. Developing Fault Models
    - 4.1. Data Requirements
    - 4.2. Module-Level Fault Models
    - 4.3. System-Level Fault Models
5. References

APPENDICES

A. Importing and Measuring a Source Repository
    - A.1. Creating a Development Environment Copy of the Repository
    - A.2. Creating a Copy of the Repository on Darwin
        - A.2.1. Creating the Repository Structure
        - A.2.2. Copying Files Into the Repository
    - A.3. Measuring a System's Evolution

B. Identifying and Counting Repaired Faults
    - B.1. LogSpecifiedVersionDiffs.pl
    - B.2. strip.pl
    - B.3. bagdiff4.pl
    - B.4. final.pl
    - B.5. extract.pl
    - B.6. CPPLexer.pm

Papers and Presentations Resulting From This CI

California Institute of Technology

[Nik03] A. Nikora, J. Munson, "The Effects of Fault Counting Methods on Fault Model Quality", submitted to the 2003 International Symposium on Software Reliability Engineering (ISSRE2003).

[Ammar03] K. Ammar, T. Menzies, A. Nikora, "How Simple is Software Defect Detection?", submitted to the 2003 International Symposium on Software Reliability Engineering (ISSRE2003).

[Nik03a] A. Nikora, J. Munson, "Predicting Fault Content for Evolving Software Systems", presented at 2003 Assurance Technology Symposium.

[Nik03b] A. Nikora, J. Munson, "Developing Fault Models for Space Mission Software", to be presented at 2003 JPL-sponsored International Conference on Space Mission Challenges for Information Technology, July 13-16, Pasadena, CA.

[Nik03c] A. Nikora, J. Munson, "Understanding the Nature of Software Evolution", to appear, proceedings of the 2003 International Conference on Software Maintenance (ICSM2003).

[Nik03d] A. Nikora, J. Munson, "Developing Fault Predictors for Evolving Software Systems", to appear, proceedings of the 2003 International Metrics Symposium (Metrics2003).

[Mun02] J. Munson, A. Nikora, "Toward A Quantifiable Definition of Software Faults", proceedings of the International Symposium on Software Reliability Engineering, Annapolis, MD, November 12-15, 2002

Papers and Presentations Resulting From This CI

California Institute of Technology

[Schn02] N. Schneidewind, "Requirements Risk versus. Reliability", presented at the International Symposium on Software Reliability Engineering, Annapolis, MD, November 12-15, 2002

[Schn02a] N. Schneidewind, "An Integrated Failure Detection and Maintenance Model", presented at the 8th IEEE Workshop on Empirical Studies of Software Maintenace (WESS), October 2, 2002, Montreal, Quebec, Canada

[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

Papers and Presentations Resulting From This CI

California Institute of Technology

[Schn01a] N. Schneidewind, "Modelling the Fault Correction Process", proceedings of the International Symposium on Software Reliability Engineering, Hong Kong, November 27-30, 2001