

---

# *A Software Measurement and Fault Modeling Technique*

**JPL IT Symposium  
November 4, 2002**

Allen Nikora  
Quality Assurance Section  
Jet Propulsion Laboratory

John C. Munson  
Department of Computer Science  
University of Idaho

The work described in this paper 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
- Measuring Software Evolution
- Measuring Software Faults
- The DARWIN Network Appliance
- Current Work
- Summary
- Future Work

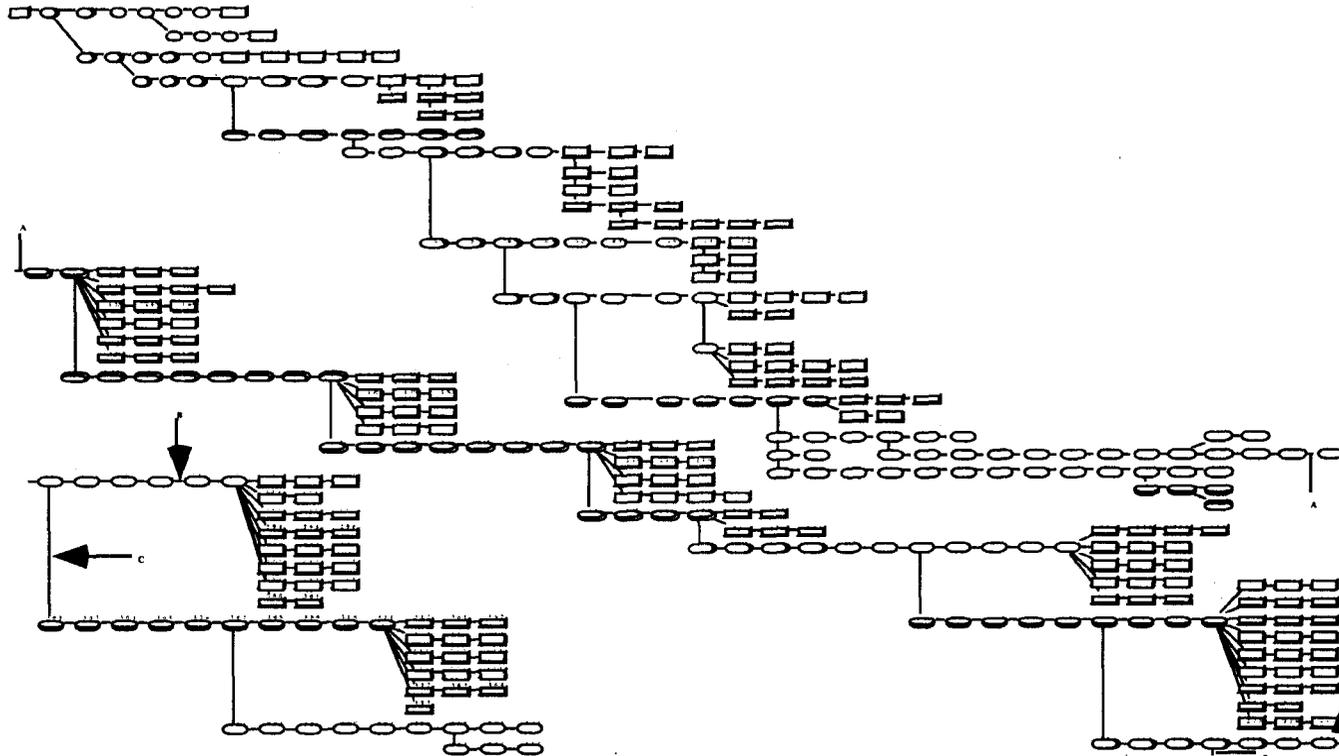
# Overview

- Classical software reliability modeling techniques can be useful in managing testing resources by answering the following questions:
  - What is the current software reliability?
  - How much longer must the software be tested to achieve the required reliability?
  - What will be the impact to a software system's reliability if insufficient testing resources are available?
- Classical software reliability models can only be used late in a software development effort, usually after unit test
- Goal – develop measurement and modeling techniques that can be used to estimate fault content/reliability prior to test

# Measuring Software Evolution

- Modern software systems change continuously
- They evolve functionally
- The code base evolves as a result
- This process must be measured to be managed

# Measuring Software Evolution



Software Evolution Tree

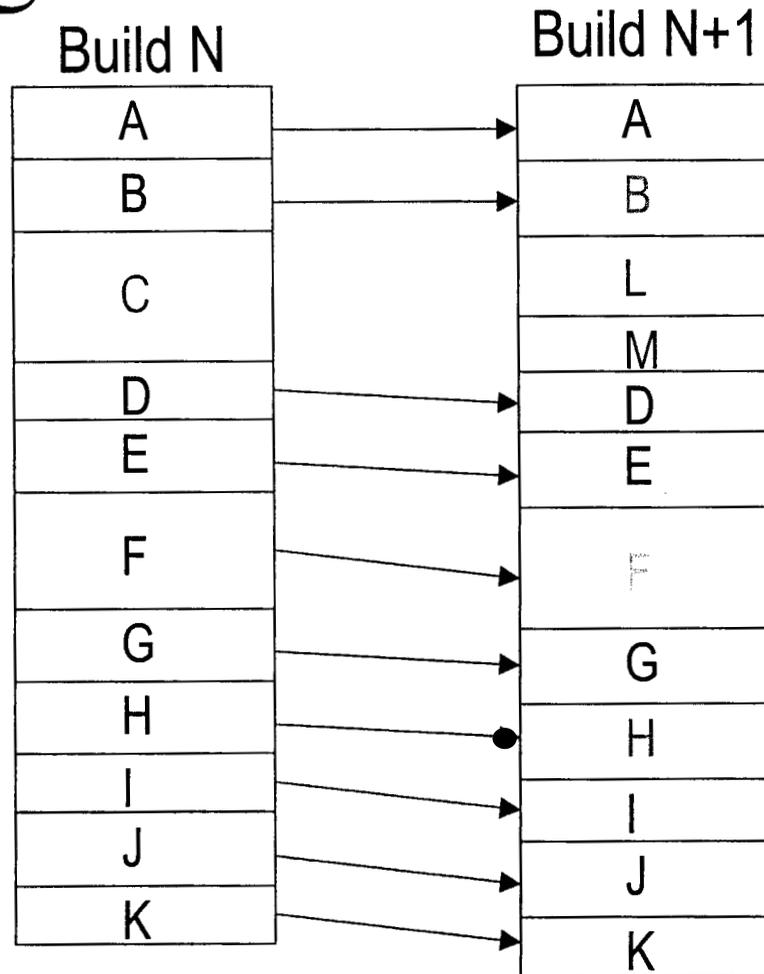
# Measuring Software Evolution

## The Measurement Problem

- Software systems are composed of components or modules
- Anytime a component changes it must be re-measured
- Over time components
  - Are added
  - Are deleted

# Measuring Software Evolution

The  
Problem:  
Part 1

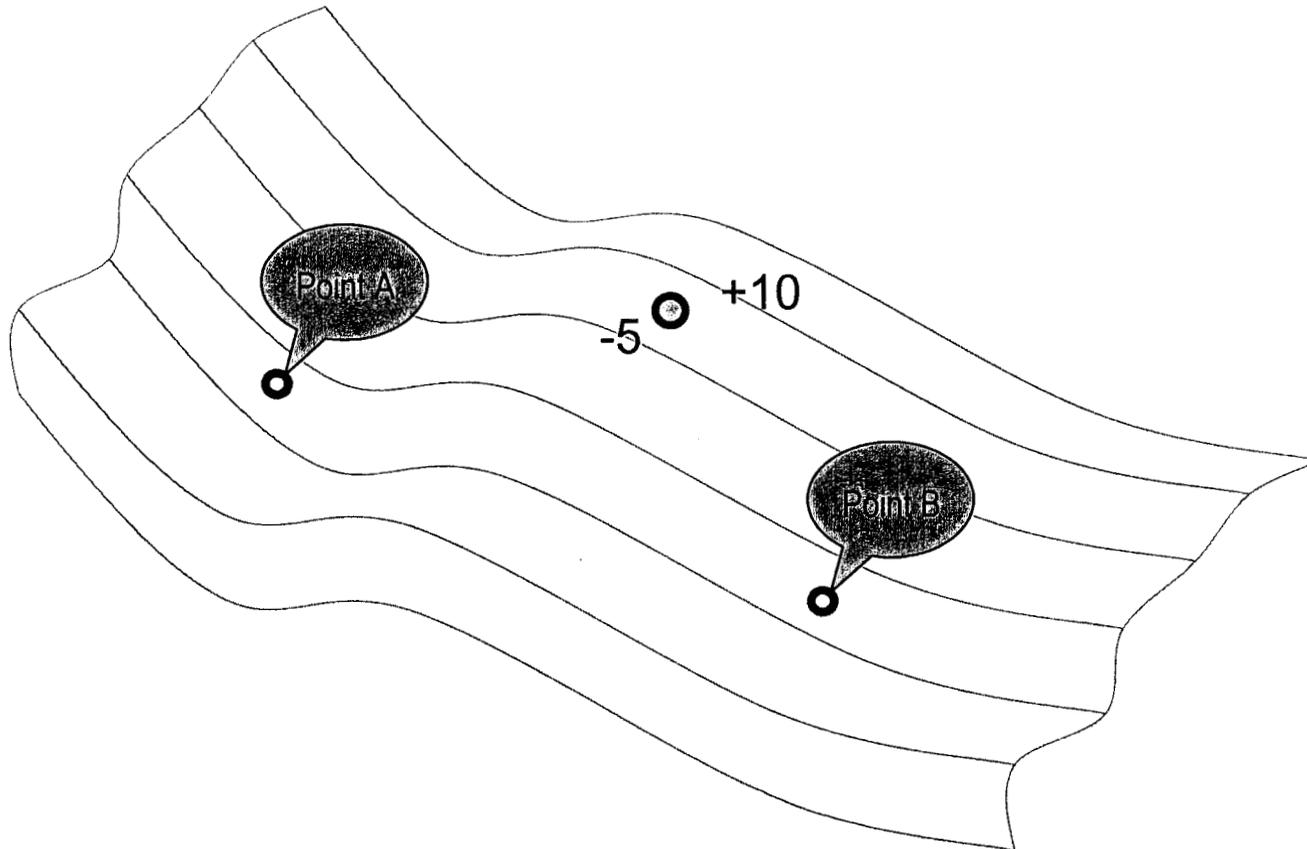


# Measuring Software Evolution

The  
Problem:  
Part 2

	Build 1		Build 2	
Module	A	B	A	B
LOC	200	250	210	230
Unique Operators	20	15	19	18

# Measuring Software Evolution



Measurement Baseline

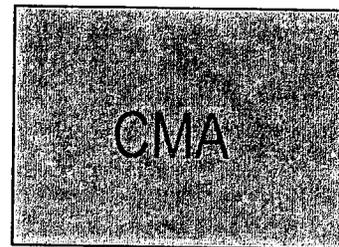
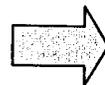
# Measuring Software Evolution

Source Code

```

{
  int i,j;
  for (i=0; Array[i][0]!='\0'; i++)
  {
    j = strcmp(String, Array[i]);
    if (j>0)
      continue;
    if (j<0)
      return -1;
    else
      return i;
  };
  return -1;
}
    
```

Module



Measurement Tool



LOC	14
Stmts	12
N1	30
N2	23
eta1	15
eta2	12
.	
.	
.	

Module Characteristics

## Measuring Software

# Measuring Software Evolution

Rotated Component Matrix

	Component		
	1	2	3
Edges	.863	.331	.267
Nodes	.860	.334	.269
Maximum path length	.858	.361	.293
Average path length	.851	.353	.331
Cycles	.690	.182	-7.059E-03
Nonexecutable statements	.645	.510	.199
Executable statements	.589	.492	.481
Unique operands	.333	.896	7.802E-02
Unique actual operands	.333	.896	7.802E-02
Comments	.251	.686	9.137E-02
Unique operators	.495	.675	9.615E-02
Paths	.173	-.118	.888
Total operators	.283	.600	.676
Total operands	.283	.600	.676
Eigenvalue:	4.903	4.279	2.342

Extraction Method: Principal Component Analysis. Rotation Method: Varimax with Kaiser Normalization.

a Rotation converged in 5 iterations.

## The Principal Components

# Measuring Software Evolution

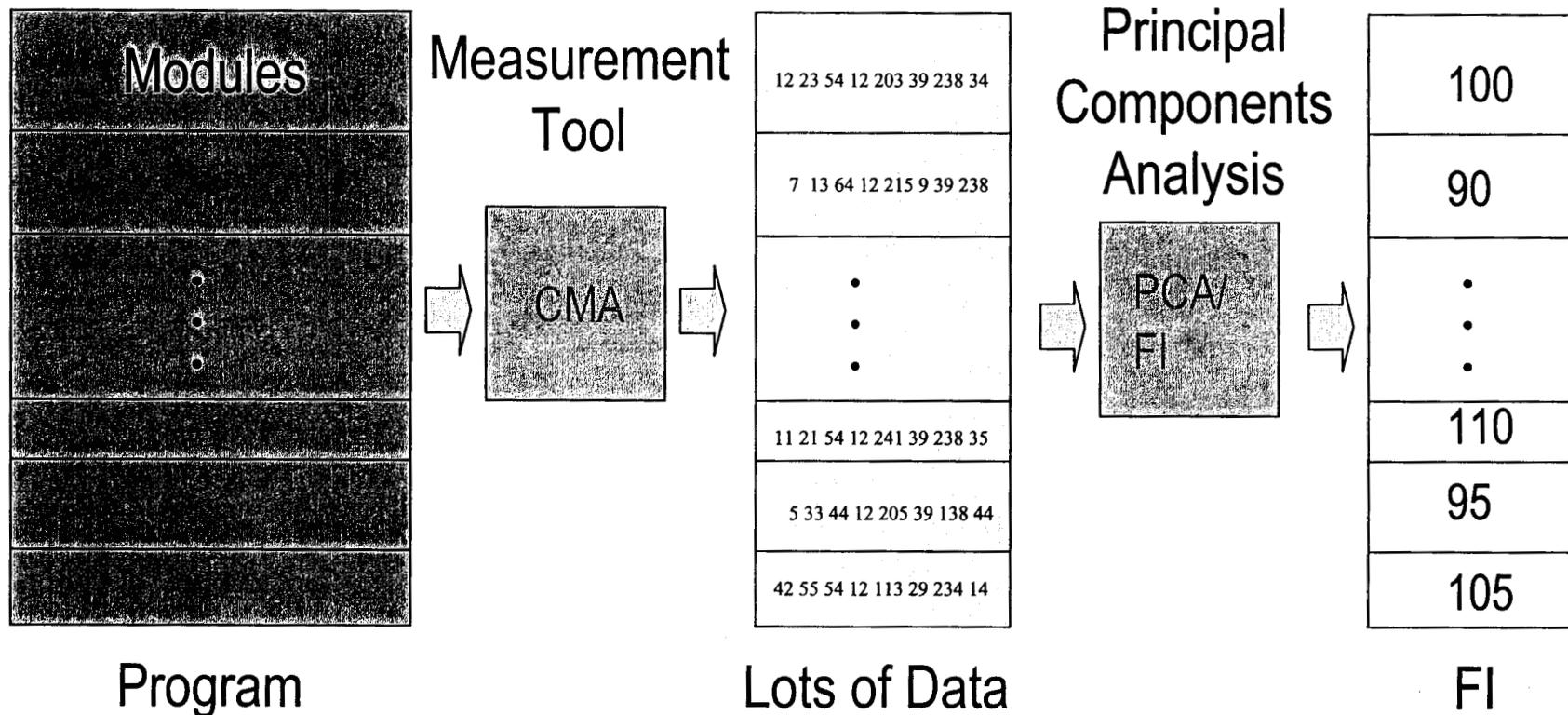
## A Fault Index

- FI is a synthesized metric

$$\rho_i^B = \sum_{j=1}^m \lambda_j^B d_j^B$$

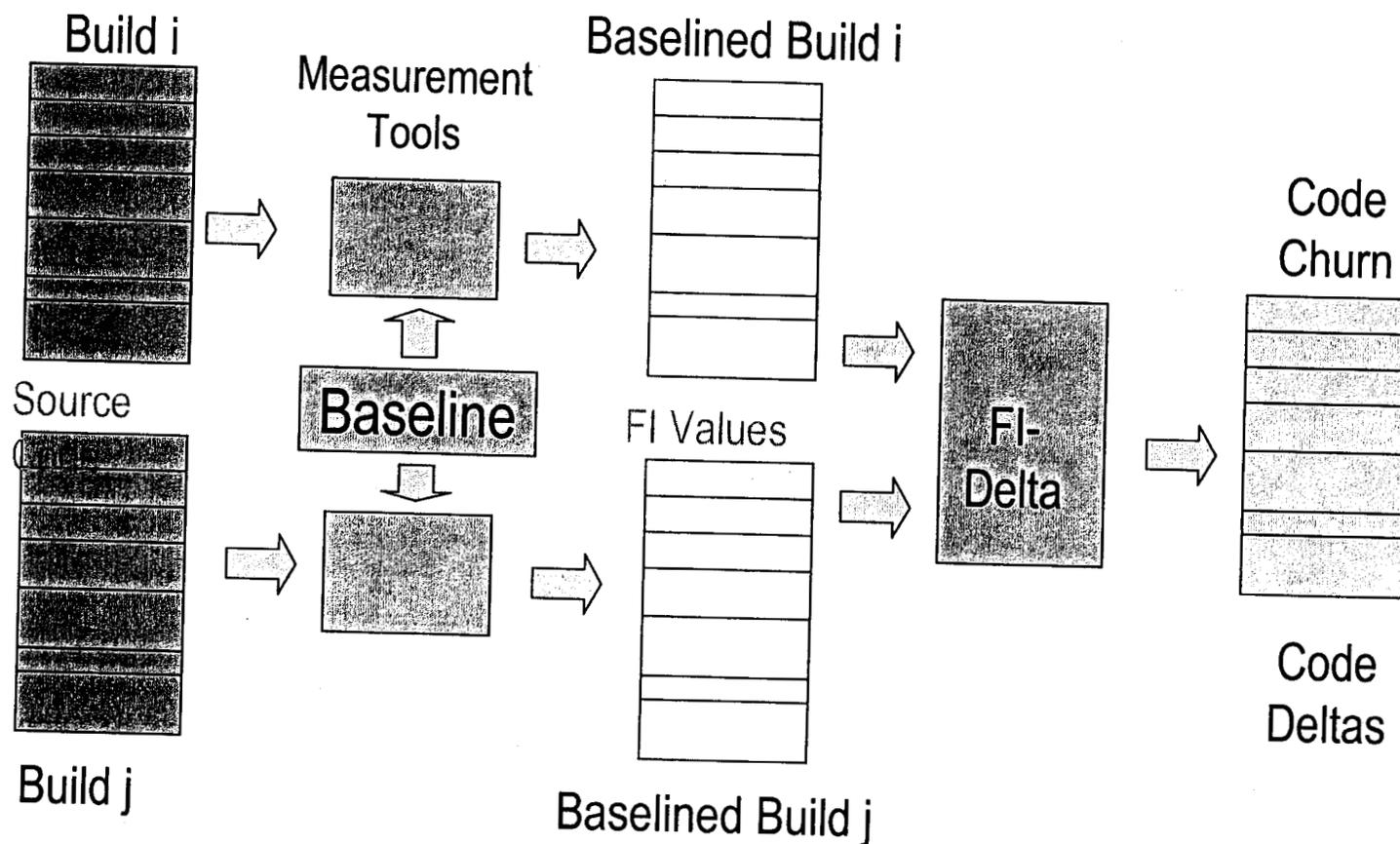
- FI is a fault surrogate
  - Composed of metrics closely related to faults
  - Highly correlated with faults

# Measuring Software Evolution



Converting Data to Information

# Measuring Software Evolution



## Comparing Two Builds

# Measuring Software Evolution

## Build Evolution

- Different modules in different builds
  - $M_a^{i,j}$  set of modules in build  $i$
  - $M_b^{i,j}$  set of modules in build  $j$
  - $M_c^{i,j}$  set of common modules in both builds

- FI of build  $i$  
$$R^i = \sum_{c \in M_c} \rho_c^i + \sum_{a \in M_a} \rho_a^i$$

- FI of build  $j$  
$$R^j = \sum_{c \in M_c} \rho_c^j + \sum_{b \in M_b} \rho_b^j$$

# Measuring Software Evolution

## Measuring Change Activity

- Code Churn

$$\chi_a^{i,j} = |\delta_a^{i,j}| = |\rho_a^{B,j} - \rho_a^{B,i}|$$

- Net Code Delta

$$\Delta^{i,j} = \sum_{c \in M_c} \delta_c^{i,j} - \sum_{a \in M_a} \rho_a^i + \sum_{b \in M_b} \rho_b^j$$

- Net Code Churn

$$\nabla^{i,j} = \sum_{m_c \in M_c} \chi_c^{i,j} + \sum_{m_a \in M_a^{i,j}} \rho_a^{B,i} + \sum_{m_b \in M_b^{i,j}} \rho_b^{B,j}$$

# Measuring Software Faults

- 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

# Measuring Software Faults

## 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

# Measuring Software Faults

## Approach (cont'd)

### Example 1

- Original statement:  $a = b + c * d$ ;
- Intended statement:  $a = b + c / d$ ;
- One token changed – “\*”  $\Rightarrow$  “/”
  - Coding error
- Count number of faults as 1

### Example 2

- Original statement:  $a = b + c * d$ ;
- Intended statement:  $a = b + (c * x) + \sin(z)$ ;
- Substantial difference between first and second statements
  - Reflects design rather than coding problem
- Fault measurement method should reflect the degree of change

# Measuring Software Faults

- 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

# Measuring Software Faults

## More Examples

### 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

### Example 2

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

### Example 3

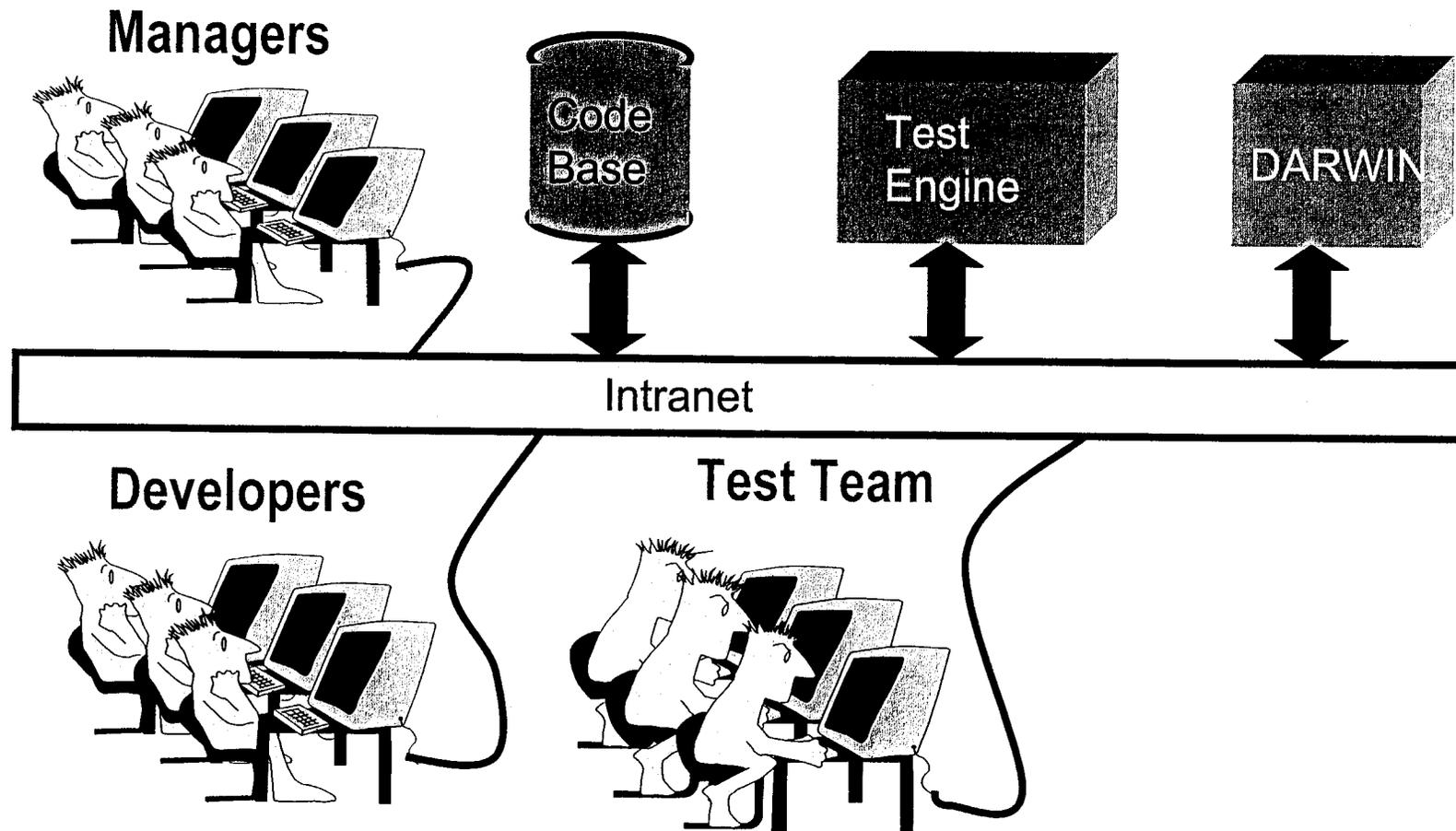
- Original statement:  $a = b - c$ ;
  - $B_3 = \{ \langle a \rangle, \langle = \rangle, \langle c \rangle, \langle - \rangle, \langle b \rangle \}$
- Modified statement:  $a = 1 + c - b$ ;
  - $B_4 = \{ \langle a \rangle, \langle = \rangle, \langle 1 \rangle, \langle + \rangle, \langle c \rangle, \langle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

# The Real Problem is Data Management

# The DARWIN Network Appliance

- Measurement process is transparent
  - incorporated in configuration control
- Measurement data converted to management information
- Designed to a measurement standard
- Monitors
  - Software evolution
  - Test activity
  - Requirements traceability

# The DARWIN Network Appliance



# The DARWIN Network Appliance

<b><i>Comm</i></b>	Total comment count
<b><i>ExStmt</i></b>	Executable statements
<b><i>NonEx</i></b>	Non executable statements
$N_1$	Total number of operands
$\eta_1$	Unique operands
$N_2$	Total number of operators
$\eta_2$	Unique operators
$\eta_3$	Unique operators with overloading
<b><i>Nodes</i></b>	Number of nodes in the module control flowgraph
<b><i>Edges</i></b>	Number of edges in the module control flowgraph
<b><i>Paths</i></b>	Number of distinct paths in the module control flowgraph
<b><i>MaxPath</i></b>	Maximum path length in the module control flowgraph
$\overline{Path}$	Average path length in the module control flowgraph
<b><i>Cycles</i></b>	Total cycle count in the module control flowgraph

## The CMA Metric Tool

# The DARWIN Network Appliance

## The PCA/FI Tool

- Builds baseline measurements
- Conversion process is transparent
  - integrated into build process
- Builds baseline transformation elements
- Converts new module measurements to baselined measures
  - Can convert one module
  - Can convert whole build

# The DARWIN Network Appliance

## The Evolution Tool

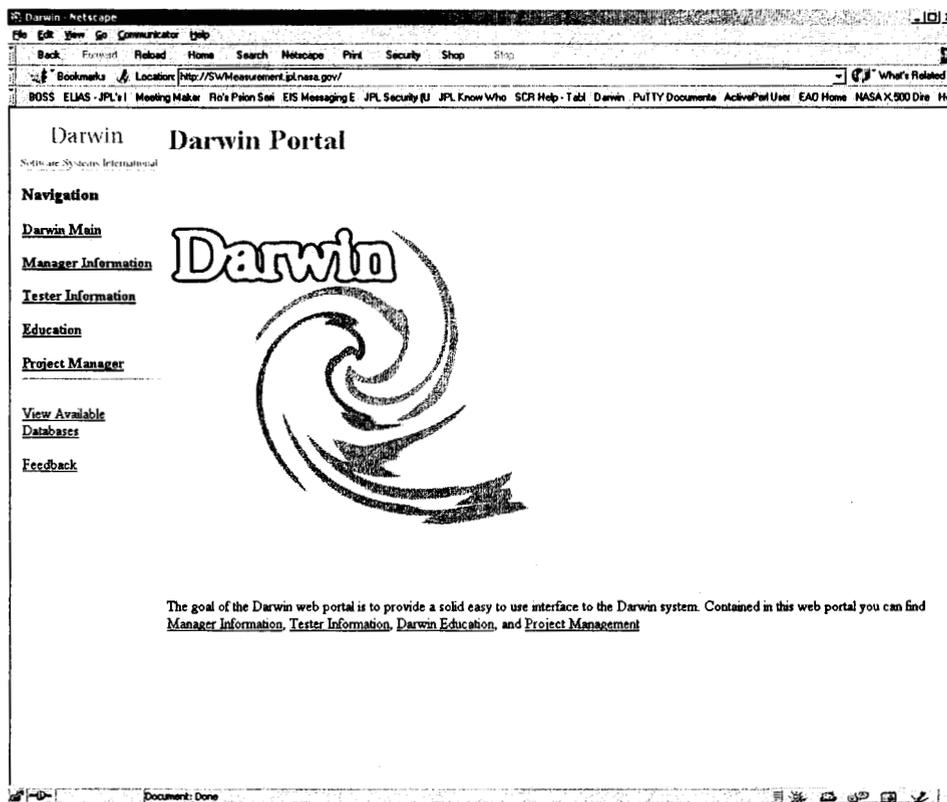
- Compares two builds
  - module by module
  - relative to baseline system
- Transparent tool
  - integrated into the build process
- Computes
  - code deltas
  - net code change (code churn)

# The DARWIN Network Appliance

## The Fault Measurement Tool

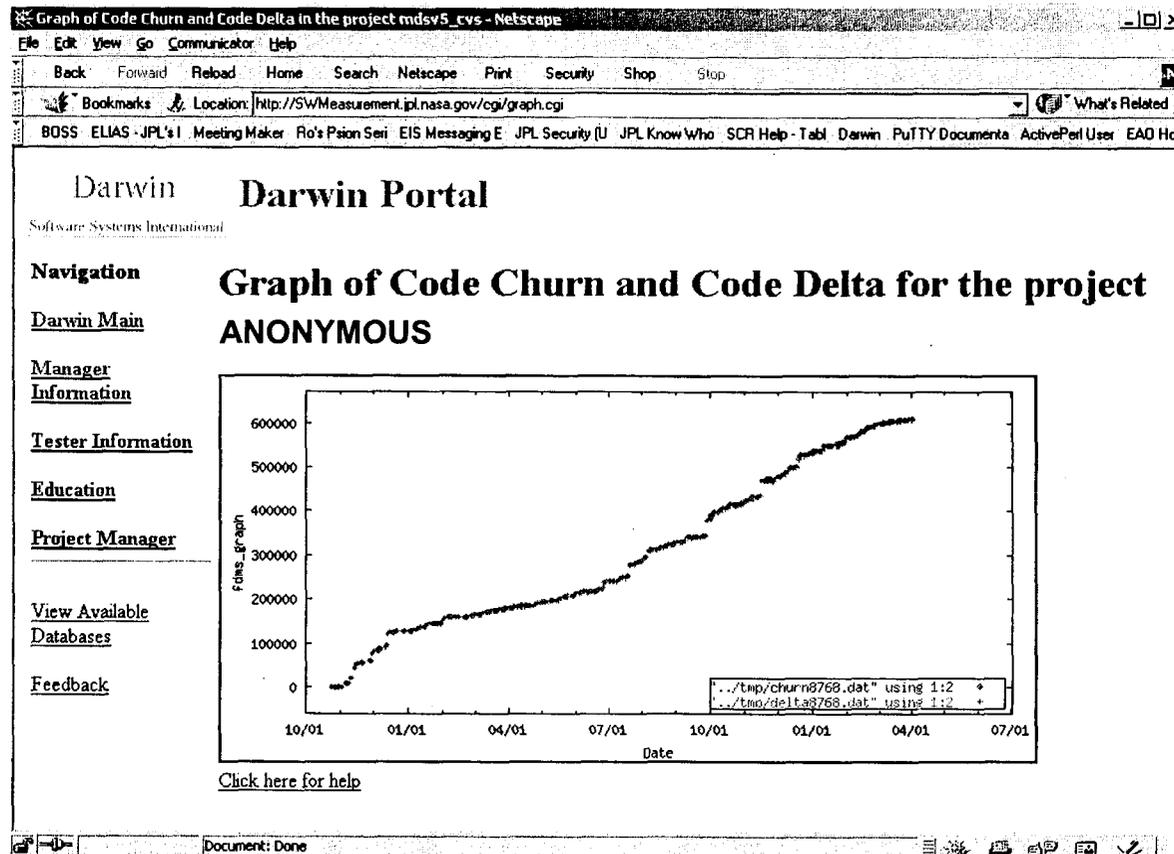
- Retrieves code deltas from CVS, RCS or SCCS
- Matches code deltas with trouble reports
- Measures associated faults

# The DARWIN Network Appliance



DARWIN Main Page  
SWMeasurement.jpl.nasa.gov (JPL internal only)

# The DARWIN Network Appliance



## DARWIN Plot of Code Churn and Code Delta

# The DARWIN Network Appliance

```

(Non-zero) Modules for build 2002-04-02 of project ANONYMOUS , sorted by Churn since baseline.

doProlog(XML_Parseparser,constENCODING*enc,constchar*s,constchar*end,inttok,constchar*next,constchar**nextPtr)
TestIntervallic:TestIntervallic()
test2st()
storeAttr(XML_Parseparser,constENCODING*enc,constchar*attStr,TAG_NAME*tagNamePtr,BINDING**bindingsPtr)
processMeasurementAndPredict(constMds:Fw.Time:Tmgt:RTepoch&current,constMds:Fw.Time:Tmgt:RTepoch&stop)
carman(intarg,char*argv[])
PREFIX(scanAttr)
Tester()
ParachuteEstimator(Traits:Thread:predictState)
dtdCopy(DTD*newDtd,constDTD*oldDtd)
*getContext(XML_Parseparser)
Parameter_getTypeFromStringNoNS(conststd:string&t)
Parameter_getTypeFromString(conststd:string&typ)
getSixDOF(constMds:Fw.Time:Tmgt:RTepoch&time)
parsePseudoAttribute(constENCODING*enc,constchar*ptr,constchar*end,constchar**namePtr,constchar**nameEndPtr,constchar**valPtr,constchar**nextTokPtr)
multiplication(ostream&error)
doContent(XML_Parseparser,intstartTagLevel,constENCODING*enc,constchar*s,constchar*end,constchar**nextPtr)
PREFIX(contentTok)
add_graph(conststd:string&spec)
TestDiscrete:TestDiscrete()
examples()
PREFIX(prologTok)
doParseXmlDecl(constENCODING*(encodingFinder)
CarEwsCommands:createXAInterceptorLinkInstance(constMds:Fw.Ews:HttpRequest&req)
cartes2sphere(Dimdm,constMds:Fw.Math:Db13Vec&cartesPos,constMds:Fw.Math:Db13Vec&cartesVel,constMds:Fw.Math:Db13Vec&cartesAcc,double*rv)
coarseNear(constMds:Fw.Math:Db13Vec&pos,Mds:Fw.Math:Db13Vec&near,double&lambda)
nearpoint(ostream&error)
  
```

	Churn From Baseline
	3243.906078
	3135.387632
	3108.478289
	3025.659494
	3011.455888
	2981.802738
	2848.905049
	2838.428107
	2823.947475
	2810.244192
	2775.849448
	2726.351943
	2724.884202
	2117.817370
	1196.440052
	806.102760
	709.630505
	691.654607
	690.016578
	670.521970
	660.459373
	610.636144
	609.995906
	570.089463
	545.460083
	415.293089
	414.782196

## Build Measurement Details

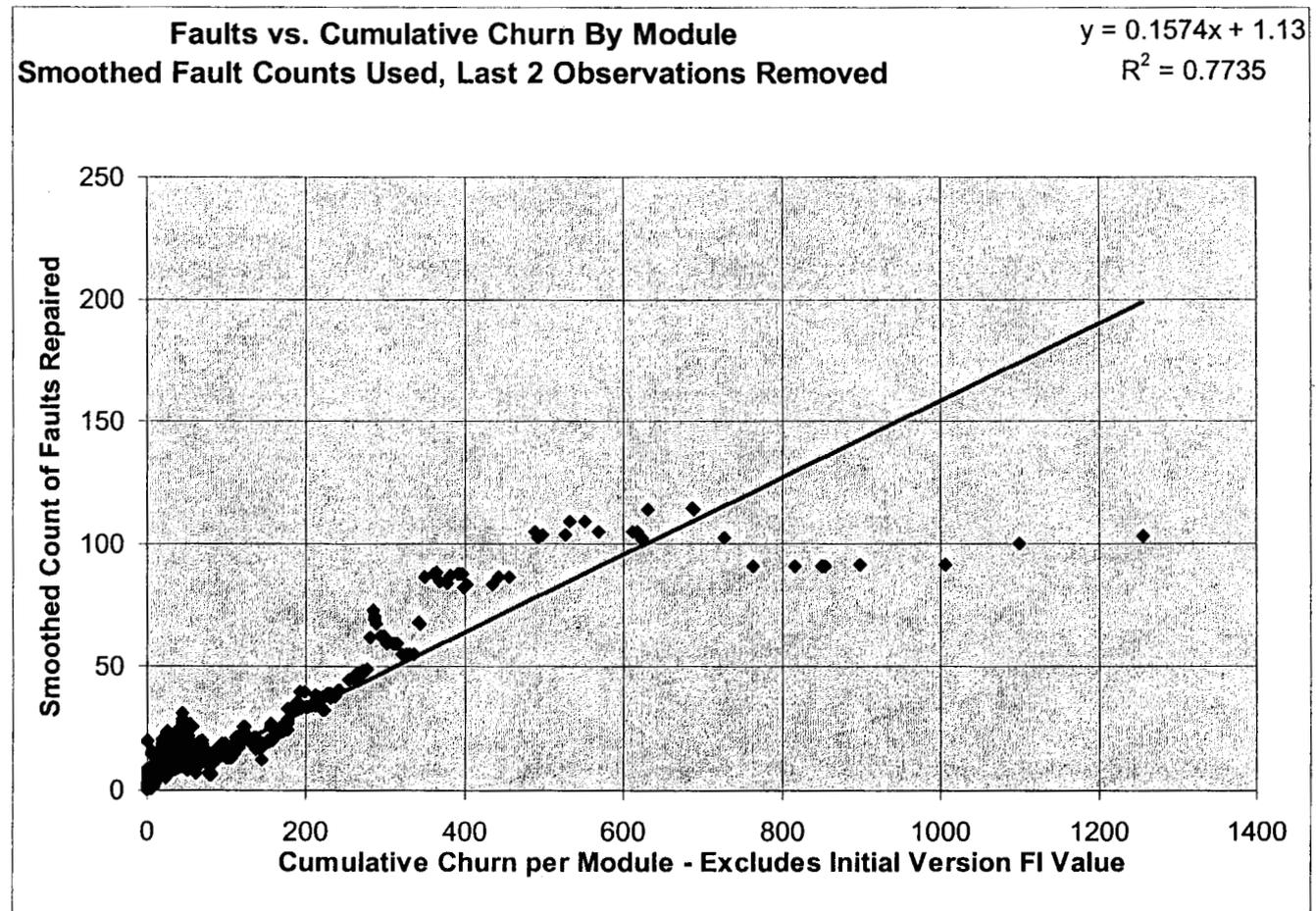


## Current Work

- Collaboration with JPL software development efforts:
  - Measure software evolution
  - Measure software faults
  - Develop fault models

# Current Work

## Fault Model Example



# Summary

- Developed a practical software measurement and fault modeling framework
  - Repeatable, consistent measurement
  - Faults measured at same level at which structural measurements are taken, i.e., function and method level
  - Easily automated
  - Transparent to developers
    - No additional activities for developers
    - No footprint in development environment
  - Dual use:
    - Resource for production software development efforts
    - Research tool

# Future Work

- Disseminate techniques
  - Other JPL projects
  - Other NASA centers
    - Funded to collaborate with GSFC SATC in FY'03
  - Industry
- Extend techniques into earlier life cycle activities