Predicting Fault Content for Evolving Software Systems

Allen P. Nikora
Jet Propulsion Laboratory,
California Institute of Technology
Pasadena, CA
Allen.P.Nikora@jpl.nasa.gov

John C. Munson
Computer Science Department
University of Idaho
Moscow, ID
jmunson@cs.uidaho.edu

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).

California Institute of Technology

Outline

- Motivation
- Measuring Structural Evolution
- Identifying and Counting Faults
- Modeling Fault Content
- On-Going Work
- Discussion and Conclusions
- References and Further Reading

Goal: Improve Understanding of Fault Generation Process

- Look for relationships between:
  - Measurements of a system's structural evolution.
  - Number of faults repaired.
- Develop software fault models
  - Accurate, consistent, repeatable measurement of structural evolution.
  - Quantifiable definition of what constitutes a fault.
- Define measurement process
  - Repeatable, accurate count of faults
  - Measure structural evolution and faults at same level (e.g., individual functions and methods)
  - Easily automated, minimal or no visible impact on developers.

California Institute of Technology

Measuring Structural Evolution

- Measure change activity between successive versions of the system.
- Granularity Issues
  - Structural evolution is measured at the module level
  - Every version of every module is measured.
  - Changes between subsequent versions of a module are measured with respect to a chosen *baseline*.
- Measurements are performed automatically outside of the development environment
  - Minimize impact of measurement activities on developers.
  - Relies on read access to CM repository.
  - Accomplished with Darwin network appliance [Cyla03].

Module Attributes

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

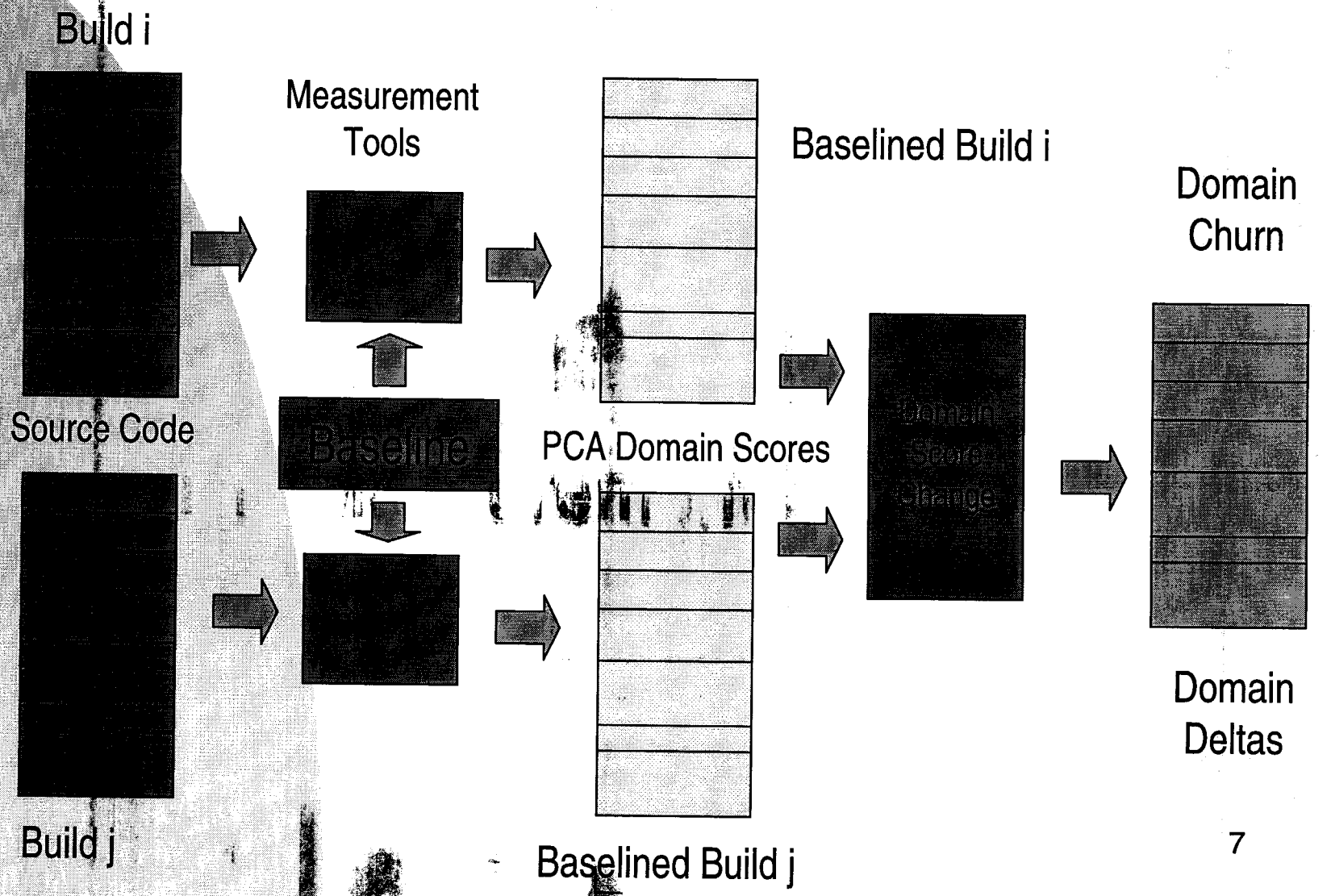
Standardized definitions were developed for each measurement – see [Cyla03]

Principal Components of Raw Metrics

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.

The Measurement Process



View of Structural Evolution at the System Level

California Institute of Technology

Graph of Code Churn and Code Delta in th...

Darwin

Software Systems International

Darwin Portal

Navigation

Darwin Main

Manager Information

Tester Information

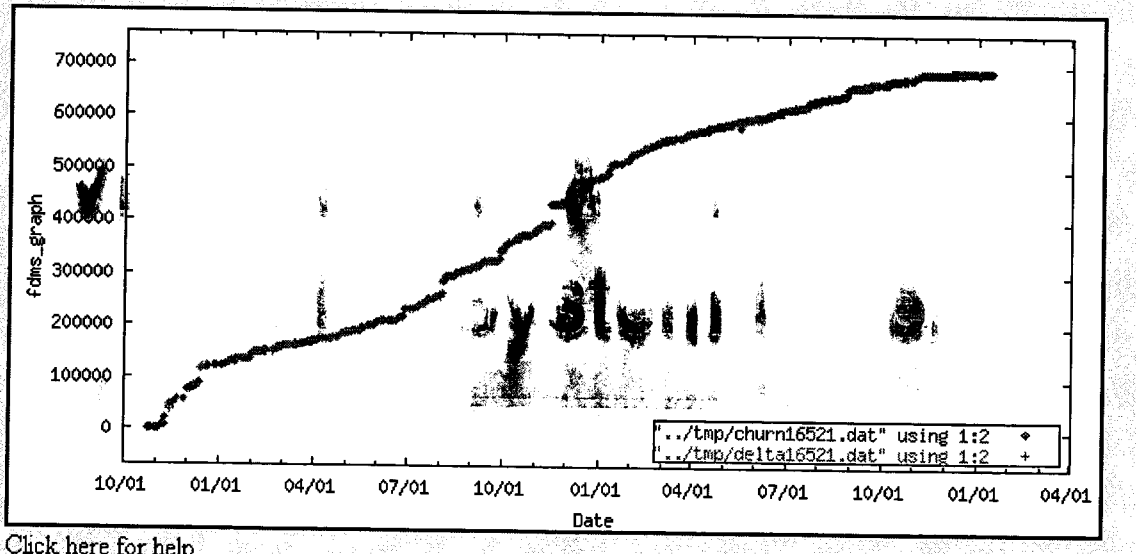
Education

Project Manager

View Available Databases

Feedback

Graph of Code Churn and Code Delta for the project fdms_project_1b.



Click here for help

documents/Metrics2003cfp.pdf

View of Structural Evolution at the Module Level

California Institute of Technology

(Non-zero) Modules for build 2001-11-02 of project project_1b, sorted by Churn since baseline.

Modulename	Churn From Baseline
doContent(XML_Parser parser, int startTagLevel, const ENCODING *enc, const char *s, const char *end, const char **nextPtr)	329.523165
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
processMeasurementAndPredict(const Mds::Fw::Time::Tmgt::RTEpoch& current, const Mds::Fw::Time::Tmgt::RTEpoch& stop)	289.353008
test2s()	279.141671
TestDiscrete::TestDiscrete()	260.394345
TestIntervallic::TestIntervallic()	256.865234
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
doTest()	237.752957
PositionEstimateFunctionTest(Dispatch& r, const std::string& key, const CGIArgs& args)	223.900440
DirectedGraph::close()	214.416659
SimpleNormalPositionEstimatorTraits::Thread::updateStateVariables(const Mds::Fw::Time::Tmgt::RTEpoch& start, const Mds::Fw::Time::Tmgt::RTEpoch& stop, const Mds::Fw::Filter::Grease::GreaseBasis& state, const Mds::Fw::Filter::Grease::GreaseBasis& covariance, const Mds::Rd::Mars::Common::SimpleAirDragModel::AirDragModelParameterType& air_drag_model_para, double spacecraft_mass, double avg_engine_thrust)	204.313726
AirDragModelParameterEstimatorTraits::Thread::predictState()	203.781925
ParachuteEstimatorTraits::Thread::predictState()	195.118089
SimpleNormalPositionEstimatorTraits::Thread::changed(const Mds::Fw::Cmp::RefCountComponentInstance monitored_sv, Mds::Fw::Dm::Vhis::ConstItemVectorRef changedItems)	182.312589
PREFIX(prologTok)	175.659645
LengthTest(Dispatch& r, const std::string& key, const CGIArgs& args)	165.230353
vxmain(int ctor, const char* argList)	164.506422
GoalNetTestHarness::createXGoalNet(Dispatch& r, const std::string& key, const CGIArgs& args)	163.479793
vxmain_internal(const char* argList)	162.467209

Identifying and Counting Faults

- Accurate software fault prediction depends on precise, measurable definition of a fault
- No existing definition of fault in measurable terms
  - IEEE Standards
    - IEEE Std 729-1983, "IEEE Standard Glossary of Software Engineering Terminology" [IEEE83]
    - IEEE Std 982.1-1988, "IEEE Standard Dictionary of Measures to Produce Reliable Software" [IEEE88]
    - IEEE Std 1044-1993, "IEEE Standard Classification for Software Anomalies" [IEEE99]
  - ODC [Chil92]
  - Previous work (Annual Oregon Workshop on Software Metrics, May 11-13, 1997) [Niko97]

Fault Enumeration

- 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 [Muns02]

Fault Enumeration (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

Fault Enumeration: Example

- 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

Fault Enumeration: Example

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

California Institute of Technology

Modeling Fault Content

- 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 [Niko03].

Modeling Results

- 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

On-Going and Future Work

- Determine whether the fault insertion rate is the same over time.
- Identify which types of faults correspond to which types of change.
- Expand the number of projects to obtain larger measurement baseline.
  - MDS, MER, MIPL at JPL
  - Collaborating with GSFC SATC to infuse measurement techniques
- Enlarge the set of structural measurements taken by network appliance (e.g., include CK OO measures [Chid94]).
- Develop training materials for practitioners
  - Measurement user's guide
  - Measurement tutorials

On-Going and Future Work (cont'd)

- Estimate amount of noise in fault counts
  - Not all changes associated with a PR may actually be repairs
  - "Pocket PRs"
    - Not a significant issue for this development effort because of how CM is set up and discipline of development team
    - May be issue for other efforts
  - Unequal test coverage – some components may be more heavily tested, finding more faults

On-Going and Future Work (cont'd)

- Resolve known fault counting inaccuracies
  - Example 1 – adding operators/operands
    - Original faulty statement: a = b + c;
    - Repaired statement: a = b – c + d;
    - Bag difference: {<->, <d>}
    - 3 tokens added or changed, however
  - Example 2 – token reordering
    - Original faulty statement: a = b - c;
    - Repaired statement: a = c - b;
    - Bag difference: {}
    - Number of reordered tokens cannot be accurately determined

California Institute of Technology

Discussion and Conclusions

- At least for the data with which we have worked, a software component's fault burden is related to the measured amount of change during its development.
- Practical techniques for measuring structural evolution and the number of repaired faults have been developed and are available for use on "real" development efforts.
  - Allows development of fault models that can provide additional information to help decide where to deploy fault identification and repair resources (e.g., test staff, additional inspections).

References and Further Reading

[Chil92]	R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, M.-Y. Wong, "Orthogonal Defect Classification - A Concept for In-Process Measurement", IEEE Transactions on Software Engineering, November, 1992, pp. 943-946.
[Cede93]	Per Cederqvist, "Version Management with CVS for CVS 1.11.1p1", available at http://www.cvshome.org/docs/manual/
[Chid94]	S. Chidamber, C. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol. 20, no. 6, June, 1994, pp. 476-493.
[Cyla03]	"The Darwin Software Engineering Measurement Appliance", Cylant, http://www.cylant.com/
[Dvo99]	D. Dvorak, R. Rasmussen, G. Reeves, A. Sacks, "Software Architecture Themes In JPL's Mission Data System", AIAA Space Technology Conference and Exposition, September 28-30, 1999, Albuquerque, NM.
[Ghok97]	S. S. Gokhale, M. R. Lyu, "Regression Tree Modeling for the Prediction of Software Quality", proceedings of the Third ISSAT International Conference on Reliability and Quality in Design, pp 31-36, Anaheim, CA, March 12-14, 1997
[Hall00]	G. A. Hall and J. C. Munson, "Software evolution: code delta and code churn", Journal of Systems and Software 54 (2) (2000) pp. 111-118
[Hunt02]	J. Hunt, W. Tichy, "Extensible Language-Aware Merging", proceedings of the 2002 International Conference on Software Maintenance, Montréal, Canada, Oct 3-6, 2002, pp 511-520

References and Further Reading (cont'd)

California Institute of Technology

[IEEE83]	"IEEE Standard Glossary of Software Engineering Terminology", IEEE Std 729-1983, Institute of Electrical and Electronics Engineers, 1983.
[IEEE88]	"IEEE Standard Dictionary of Measures to Produce Reliable Software", IEEE Std 982.1-1988, Institute of Electrical and Electronics Engineers, 1989.
[IEEE93]	"IEEE Standard Classification for Software Anomalies", IEEE Std 1044-1993, Institute of Electrical and Electronics Engineers, 1994.
[Khos01]	T. Khoshgoftaar, "An Application of Zero-Inflated Poisson Regression for Software Fault Prediction", proceedings of the 12th International Symposium on Software Reliability Engineering, pp 66-73, Hong Kong, Nov, 2001.
[Khos01a]	T. M. Khoshgoftaar, E. B. Allen, "Modeling Software Quality with Classification Trees", in H. Pham (ed), Recent Advances in Reliability and Quality Engineering, Chapter 15, pp 247-270, World Scientific Publishing, Singapore, 2001.
[Muns90]	J. C. Munson and T. M. Khoshgoftaar, "Regression Modeling of Software Quality," Information and Software Technology, Vol 32 No. 2 March 1990, pp. 105-114.
[Muns98]	J. Munson and A. Nikora, "Estimating Rates Of Fault Insertion And Test Effectiveness In Software Systems" Proceedings of the Fourth ISSAT International Conference on Reliability and Quality in Design, August 12-14, 1998 pp. 263-269.
[Muns02]	J. Munson, A. Nikora, "Toward a Quantifiable Definition of Software Faults", Proceedings of the 13th IEEE International Symposium on Software Reliability Engineering, IEEE Press.

References and Further Reading (cont'd)

California Institute of Technology

[Muns03]	J. Munson, Software Engineering Measurement, CRC Press, 2003, ISBN 0849315034 .
[Niko97]	A. Nikora, J. Munson, "Finding Fault with Faults: A Case Study", with J. Munson, proceedings of the Annual Oregon Workshop on Software Metrics, Coeur d'Alene, ID, May 11-13, 1997.
[Niko98]	A. P. Nikora, J. C. Munson, "Determining Fault Insertion Rates For Evolving Software Systems", proceedings of the 1998 IEEE International Symposium of Software Reliability Engineering, Paderborn, Germany, November 1998, IEEE Computer Society Press.
[Niko01]	A. Nikora, J. Munson, "A Practical Software Fault Measurement and Estimation Framework", Industrial Presentations proceedings of the 12th International Symposium on Software Reliability Engineering, Hong Kong, Nov 27-30, 2001.
[Niko03]	A. Nikora, J. Munson, "Developing Fault Predictors for Evolving Software Systems", to appear in the proceedings of the 9th International Symposium on Software Metrics (Metircs2003), Sydney, Australia, Sept 3-5, 2003
[Schn97]	N. F. Schneidewind, "Software Metrics Model for Integrating Quality Control and Prediction", proceedings of the 8th International Symposium on Software Reliability Engineering, pp 402-415, Albuquerque, NM, Nov, 1997.
[Schn01]	N. F. Schneidewind, "Investigation of Logistic Regression as a Discriminant of Software Quality", proceedings of the 7th International Software Metrics Symposium, pp 328-337, London, April, 2001.