

# Invitation to ICS 691: Automated Software Engineering Measurement and Analysis

(Shameless commercial plug)

Philip Johnson  
Collaborative Software Development Laboratory  
Information and Computer Sciences  
University of Hawaii  
Honolulu HI 96822

(1)

## SE Motherhood and Apple Pie

Things like the following are "good to know":

- How long it takes you to build software
- Whether your code is currently flaky or not
- The kinds of mistakes you make
- How long it might be to ship the next increment
- What would be the most important thing to work on next

The question is (to paraphrase David Byrne):

- "How do we get there?"

(2)

## In the beginning, there was...

Table C37a Test Report Template		Table C47a Task Planning Template											
Student	Jane Doe	Student										Date	
Instructor	Philip Johnson	Instructor											
Test Name	Input test	Project											
Test Objective	Test input under												
Test Description	Test mean and std	Task	Hours	Planned Value	Completed Hours	Cumulative Planned Value	Due Monday	Due	Error Value	Completed Error Value			
Test Conditions	None												
Expected Results	Mean should be												
Actual Results	Mean @ 5, std.												
Projected LOC:	$P = BA/NQ$												
Correlation (prior planned vs. actual)	$r^2$												
Regression Parameter:	$\alpha$ (use 0 if $r^2 < 0.50$ )												
Regression Parameter:	$\beta$ (use 1 if $r^2 < 0.50$ )												
Estimated New and Changed LOC:	$N = \alpha + \beta(P-M)$												
Estimated Total LOC:	$T = N + B - D - M + R$												
Estimated Total New Resized sum of * LOC:													
Prediction Range:	Range	Defect Removal Efficiency	Plan	Actual	To Date								
Upper Prediction Interval/Largest Likely Size	UPI	Defects/Hour - Design review											
Lower Prediction Interval/Smallest Likely Size	LPI	Defects/Hour - Code review											
Prediction Interval Percent:	70% or	Defects/Hour - Compile											
		Defects/Hour - Test											
		DRI/(Code Review)											
		DRI/(Code Review/UT)											
		DRI/(Compile/UT)											

(3)

## Generation 1: PSP (1995)

Characteristics:

- Tool support minimal/discouraged.
- Developer-initiated data collection.
- Developer-initiated analysis.
- Any data can be collected and analyzed.

Results from our case studies:

- Complicated to learn
- Low adoption, high motivational overhead
- QA on data very difficult

Might manual data collection and analysis be the problem?

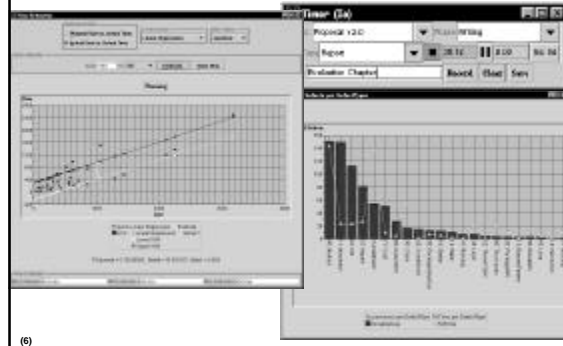
(4)

## Generation Gaps

	Gen. 1 PSP		
Developer-initiated collection	Yes		
Developer-initiated analysis	Yes		
Collection overhead	High		
Analysis overhead	High		
Data quality assurance overhead	High		
Data choice flexibility	High		
Adoption overhead	Very High		

(5)

## So then CSDL made tools...



(6)

## Generation 2: Leap (1998)

### Characteristics:

- Partially automated data collection
- Totally automated data analysis
- Developer-initiated data collection
- Developer-initiated data analysis
- Limitations on data collected/analyzed.

### Results from our case studies:

- Some PSP analyses may be unnecessarily complex.
- Adoption easier, but still low

Is overhead of "initiation" alone problematic?

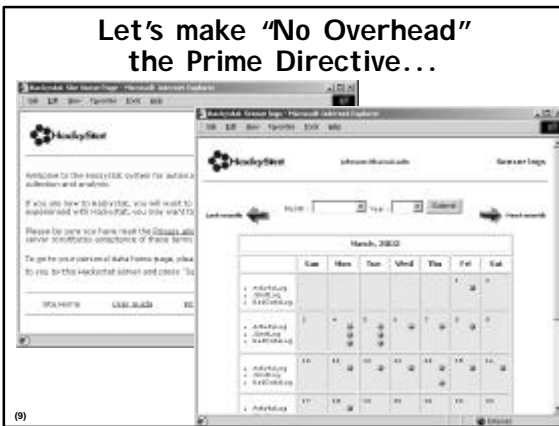
(7)

## Generation Gaps

	Gen. 1 PSP	Gen. 2 Leap	
Developer-initiated collection	Yes	Yes	
Developer-initiated analysis	Yes	Yes	
Collection overhead	High	Medium	
Analysis overhead	High	Low	
Data quality assurance overhead	High	Medium	
Data choice flexibility	High	Medium	
Adoption overhead	Very High	High	

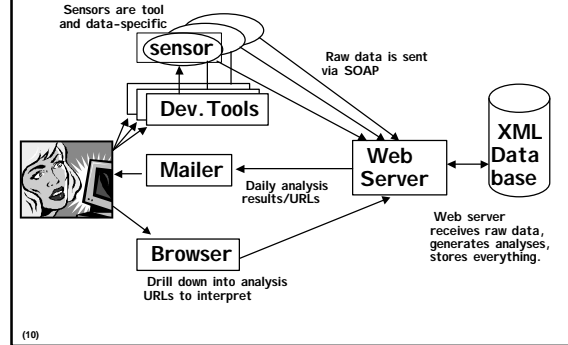
(8)

## Let's make "No Overhead" the Prime Directive...



(9)

## Hackstat Architecture



(10)

## Hackstat: Third generation

### Characteristics:

- Web services architecture:
  - Sensors attached to development tools
  - Sensors send raw data to web server
  - Web server analyzes data, emails results to developer.
- "Overhead" limited to installation/configuration.
  - Daily collection/analysis completely automated.
- Data limited in form and content
  - must be observed via tool instrumentation

(11)

## Generation Gaps

	Gen. 1 PSP	Gen. 2 Leap	Gen. 3 Hackstat
Developer-initiated collection	Yes	Yes	No
Developer-initiated analysis	Yes	Yes	No
Collection overhead	High	Medium	None
Analysis overhead	High	Low	Low/None
Data quality assurance overhead	High	Medium	None
Data choice flexibility	High	Medium	Low
Adoption overhead	Very High	High	Low

(12)

## Current status

Limited functionality version in daily use since August, 2001. (Six current users).

Sensors developed for Emacs, JBuilder, JUnit

Second architectural redesign completed.

Validation of file-level effort data collection quality scheduled for Summer, 2002.

Deployment in classroom setting scheduled for Fall, 2002.

(13)

## High-level research issues

Impact of total automation constraint:

- How useful are the resulting data/analyses?

Hackystat and XP:

- Total automation is appealing for XP.
- How to deal with pair programming?

Social and organizational impact:

- Abuse of data by management?

(14)

## For more information

<http://csdl.ics.hawaii.edu/Research/Hackystat>

- Overview of project, technical reports

<http://katrina.ics.hawaii.edu/>

- Hackystat public server

<http://csdl.ics.hawaii.edu/Tools/Hackystat/>

- Browsable design documentation
- Browsable source code
- Distribution snapshot

(15)

## So what does this have to do with ICS 691?

Course structure:

- Readings on software measurement/analysis:
  - Presented by pairs of students
  - Divided up at beginning of semester
- Personal metrics collection:
  - Install hackystat sensors
  - Use JBuilder (or Emacs) for development
- Software development
  - Hack Hackystat
  - Scratch an itch
  - Build something cool

(16)