

INVESTIGATING INDIVIDUAL SOFTWARE DEVELOPMENT: AN EVALUATION  
OF THE LEAP TOOLKIT

A DISSERTATION SUBMITTED TO THE GRADUATE DIVISION OF THE  
UNIVERSITY OF HAWAI'I IN PARTIAL FULFILLMENT  
OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

COMMUNICATION AND INFORMATION SCIENCES

AUGUST 2000

By

Carleton A. Moore

Dissertation Committee:

Philip Johnson, Chairperson

James Corbett

Elizabeth Davidson

Marie Iding

Larry Osborne

We certify that we have read this dissertation and that, in our opinion, it is satisfactory in scope and quality as a dissertation for the degree of Doctor of Philosophy in Communication and Information Sciences.

DISSERTATION COMMITTEE

---

Chairperson

©Copyright 2000

by

Carleton A. Moore

To  
Mom, Dad, and Scott

# Acknowledgments

Philip Johnson, my advisor, dissertation chair and mentor, provided amazing vision and unlimited support during this journey. I have treasured our many conversations and your enthusiasm for producing world-class software engineers.

The past and present members of CSDL, Joe Dane, Anne Disney, Tie Fang, Jennifer Geis, Monir Hodges, Mette Moffet, Danu Tjahjono, Russell Tokujama, Rosemary Sumajit, and Dadong Wan helped make work and research fun. You all gave me an excellent sounding board and helped me improve my presentations and thinking.

Mom and Dad, your love and support has made all of this possible. Thank you for encouraging my academic ambitions.

Scott Moore has shown me that it is ok to take risks and that we can follow our dreams.

Craig Chun and Tyler Tokiyoka have remained great friends, listening to my complaints and helping me get away from work when I needed it.

My fellow CIS students, you have helped me survive and grow over the last four years. Without your kind words and support, I know this would not have been possible.

I thank my committee members, James Corbett, Elizabeth Davidson, Marie Iding, and Larry Osborne. Your time and support has been priceless.

Funds for this research were provided in part by a grant from the National Science Foundation CCR 98-04010.

# Abstract

Software developers work too hard and yet do not get enough done. Developing high quality software efficiently and consistently is a very difficult problem. Developers and managers have tried many different solutions to address this problem. Recently their focus has shifted from the software organization to the individual software developer. For example, the Personal Software Process incorporates many of the previous solutions while focusing on the individual software developer.

This thesis presents the Leap toolkit, which combines ideas from prior research on the Personal Software Process, Formal Technical Review and my experiences building automated support for software engineering activities. The Leap toolkit is intended to help individuals in their efforts to improve their development capabilities. Since it is a light-weight, flexible, powerful, and private tool, it provides a novel way for developers to gain valuable insight into their own development process. The Leap toolkit also addresses many measurement and data issues involved with recording any software development process.

The main thesis of this work is that the Leap toolkit provides a novel tool that allows developers and researchers to collect and analyze software engineering data. To investigate some of the issues of data collection and analysis, I conducted a case study of 16 graduate students in an advanced software engineering course at the University of Hawaii, Manoa. The case study investigated: (1) the relationship between the Leap toolkit's time collection tools and "collection stage" errors; and (2) different time estimation techniques supported by the Leap toolkit.

The major contributions of this research includes (1) the LEAP design philosophy; (2) the Leap toolkit, which is a novel tool for individual developer improvement and software engineering research; and (3) the insights from the case study about collection overhead, collection error and project estimation.

# Table of Contents

Acknowledgments . . . . .	v
Abstract . . . . .	vi
List of Tables . . . . .	xi
List of Figures . . . . .	xii
1 Introduction . . . . .	1
1.1 Why is Quality Software Development Important? . . . . .	1
1.2 Traditional Solutions . . . . .	2
1.3 LEAP: Giving developers more control and insight . . . . .	3
1.4 Thesis Statement . . . . .	9
1.5 Investigating Individual Software Development . . . . .	9
1.6 Contributions . . . . .	10
1.7 Organization of the Dissertation . . . . .	11
2 Related Work . . . . .	12
2.1 Personal Software Process . . . . .	13
2.1.1 Purpose of the PSP . . . . .	14
2.1.2 Learning the PSP . . . . .	14
2.1.2.1 PSP0: The Baseline Process . . . . .	15
2.1.2.2 PSP1: The Personal Planning Process . . . . .	16
2.1.2.3 PSP2: Personal Quality Management . . . . .	17
2.1.2.4 PSP3: Cyclic Personal Process . . . . .	19
2.1.3 Using the PSP . . . . .	20
2.1.3.1 An Example use of the PSP . . . . .	20
2.1.4 Evaluations of the PSP . . . . .	38
2.1.5 Disney Thesis on Data Quality in the PSP . . . . .	42
2.1.6 The PSP and LEAP . . . . .	44
2.2 Automated PSP Tools . . . . .	44
2.2.1 Full automation . . . . .	44
2.2.1.1 psptool . . . . .	44
2.2.1.2 PSP Studio . . . . .	45
2.2.1.3 PSP Tool . . . . .	45
2.2.2 Partial PSP automation . . . . .	45
2.2.2.1 pplog-mode, PPLog Control, Timmie and makelog . . . . .	46
2.2.2.2 titrax . . . . .	46
2.2.2.3 PC LOC Accounting Tools . . . . .	46

2.2.2.4	locdelta . . . . .	47
2.2.2.5	LOCC . . . . .	47
2.2.3	Automated PSP tools and LEAP . . . . .	47
2.3	Formal Technical Review . . . . .	47
2.3.1	The FTR Process . . . . .	48
2.3.2	The Spectrum of Reviews . . . . .	49
2.3.3	Formal Technical Review and LEAP . . . . .	50
2.4	Measurement Dysfunction . . . . .	50
2.4.1	Measurement Dysfunction in the PSP . . . . .	51
2.4.2	Measurement Dysfunction in Review . . . . .	52
2.4.3	Measurement Dysfunction and LEAP . . . . .	54
3	Supporting Software Developer Improvement with LEAP . . . . .	55
3.1	Background . . . . .	55
3.2	LEAP Design criteria . . . . .	58
3.2.1	Criteria #1: Light-Weight . . . . .	58
3.2.2	Criteria #2: Empirical . . . . .	58
3.2.3	Criteria #3: Anti-measurement Dysfunction . . . . .	59
3.2.4	Criteria #4: Portable . . . . .	59
3.3	Leap toolkit: a reference implementation of the LEAP philosophy . . . . .	59
3.3.1	Supporting personal process improvement . . . . .	59
3.3.2	Supporting Group Review . . . . .	60
3.3.3	Providing Light-Weight Support . . . . .	61
3.3.4	Supporting Empirical Data Analysis . . . . .	61
3.3.5	Reducing Measurement Dysfunction . . . . .	61
3.3.6	Providing a Portable Tool . . . . .	62
3.4	The Leap toolkit's architecture . . . . .	62
3.4.1	Data Files . . . . .	63
3.4.2	Model-View-Controller . . . . .	67
3.4.3	Java Packages . . . . .	67
3.4.4	Generic Analyses . . . . .	69
3.4.5	Extension mechanism . . . . .	71
3.5	An Example use of the Leap toolkit . . . . .	71
3.6	Comparing the Leap toolkit example and the PSP example . . . . .	89
3.6.1	Planning new projects . . . . .	91
3.6.2	Recording process data . . . . .	92
3.6.3	Flexible process definition . . . . .	93
3.6.4	Conducting reviews . . . . .	93
3.6.5	Conducting the postmortem . . . . .	94
3.7	Benefits of Leap toolkit's design . . . . .	94
3.7.1	The Leap toolkit addresses many important classes of data errors found in the PSP . . . . .	95
3.7.2	The Leap toolkit addresses collection stage errors by reducing the overhead of data collection . . . . .	103
3.7.3	The Leap toolkit provides data analyses that are not practical with a manual method. . . . .	103



4	Investigating Individual Software Development . . . . .	104
4.1	Investigating collection stage errors . . . . .	104
4.1.1	Case Study Method . . . . .	105
4.1.1.1	Ensuring Anonymity . . . . .	105
4.1.2	Data Collection . . . . .	105
4.1.2.1	Student's raw data . . . . .	106
4.1.2.2	Surveys . . . . .	106
4.1.2.3	Survey #1 . . . . .	106
4.1.2.4	Survey #2 . . . . .	107
4.1.2.5	Survey #3 . . . . .	107
4.1.2.6	Leap Survey #4 . . . . .	107
4.1.3	Interviews with students . . . . .	108
4.1.4	Data analysis . . . . .	108
4.2	Investigating planning and estimation . . . . .	108
4.2.1	Empirical Investigation Environment . . . . .	109
4.2.2	Independent and Dependent variables . . . . .	110
4.2.3	The Design . . . . .	110
4.2.4	Analysis . . . . .	111
5	Results . . . . .	112
5.1	Leap's design as a response to user feedback . . . . .	112
5.1.1	From single to cross-platform availability . . . . .	113
5.1.2	From simple to sophisticated data collection and analysis . . . . .	114
5.1.3	From opaque to translucent data representation . . . . .	116
5.1.4	From simple to sophisticated project management . . . . .	116
5.2	Collection errors and collection overhead . . . . .	117
5.2.1	Using Naia for data entry makes data entry harder . . . . .	119
5.2.2	Reported use of Naia has a higher correlation to rounded times . . . . .	120
5.2.3	Limitations . . . . .	121
5.2.4	Summary . . . . .	122
5.3	Comparing time estimation methods . . . . .	122
5.3.1	Size estimation skills improved . . . . .	123
5.3.2	Time estimation skills improved . . . . .	124
5.3.3	Empirical Investigation . . . . .	126
5.3.3.1	Individual Student Results . . . . .	126
5.3.3.2	Class Results . . . . .	128
5.3.4	Limitations . . . . .	129
5.3.5	Summary . . . . .	129
5.4	Additional Results . . . . .	130
5.4.1	Rounding errors did not affect the accuracy of the estimates . . . . .	130
5.4.2	There was very little evidence of measurement dysfunction in the class . . . . .	131
5.4.3	Student productivity stayed steady . . . . .	132
5.4.4	Experience leads to overconfidence . . . . .	134
5.5	Summary . . . . .	135
6	Conclusion . . . . .	136
6.1	Research summary . . . . .	136

6.2	Research contributions . . . . .	138
6.2.1	LEAP design philosophy . . . . .	138
6.2.2	Leap toolkit . . . . .	139
6.2.3	Case study insights . . . . .	139
6.3	Future directions . . . . .	140
6.3.1	Automated data collection . . . . .	140
6.3.2	Agents for just in time analysis . . . . .	141
6.3.3	Database for storing the Leap data . . . . .	142
6.3.4	Additional analyses and features . . . . .	142
6.3.5	Replication of the study and the Reflective Software Engineering class . . .	143
6.3.6	Investigations into FTR and personal data collection . . . . .	143
6.4	Lessons for current software development improvement practice . . . . .	143
6.5	Implications for future software development improvement efforts . . . . .	144
A	Leap Evaluation Surveys . . . . .	146
B	Survey Data . . . . .	157
B.1	Survey # 1 . . . . .	157
B.2	Survey #2 . . . . .	162
B.3	Survey #3 . . . . .	162
B.4	Survey # 4 . . . . .	162
C	Student Interviews . . . . .	164
C.1	How do you feel about collecting size, time, and defect data about your software development process? . . . . .	164
C.2	Was collecting the Leap data easy, neutral, or hard? . . . . .	166
C.3	Did collecting all this data change the way you program? . . . . .	167
C.4	Do you think your Leap data accurately reflects what really happened? . . . . .	169
C.5	How about the size data from LOCC, do you think that it was accurate? . . . . .	172
C.6	How do you feel about making an estimate of how long a project will take? . . . .	173
C.7	Were you aware of your estimate while you were programming? . . . . .	175
C.8	How much pressure did you feel to make your actual time match your estimate? . .	176
C.9	In general, how accurate were your time estimates? . . . . .	177
C.10	Do you think your time estimating skills are improving? . . . . .	178
C.11	What caused your estimates to be wrong? . . . . .	180
C.12	Did you compare your Leap data with other students in the class? . . . . .	181
C.13	Compare your programming skills now to your programming skills before the class.	182
C.14	What is the most important thing you learned from this class? . . . . .	184
C.15	What would you change about Leap to make it better? . . . . .	186
C.16	In what situations would you use Leap in the future? . . . . .	189
	Bibliography . . . . .	192

# List of Tables

<u>Table</u>	<u>Page</u>
2.1 Cam's historical size and time data. . . . .	23
2.2 Cam's historical time distribution. . . . .	27
2.3 Cam's historical defect distribution. . . . .	28
2.4 Cam's planned defect distribution. . . . .	29
5.1 Correlation between Time, LOC, and Method estimation accuracy . . . . .	125
5.2 Student #12's relative prediction error . . . . .	127
5.3 Student #15's relative prediction error . . . . .	127
5.4 Student #4's relative prediction error . . . . .	128
5.5 Relative Prediction Error . . . . .	129
5.6 Class' estimation error . . . . .	130
B.1 Survey # 1 Results: Personal Programming . . . . .	157
B.2 Survey # 1 Results: SE Experience and Attitutdes . . . . .	158
B.3 Survey # 1 Results: Time Collection . . . . .	158
B.4 Survey # 2 Results: Leap Usability . . . . .	159
B.5 Survey # 2 Results: Time Collection . . . . .	159
B.6 Survey # 2 Results: Time Estimation . . . . .	160
B.7 Survey # 3 Results: Time Collection . . . . .	160
B.8 Survey # 3 Results: Time Estimation . . . . .	161
B.9 Survey # 3 Results: Defect Collection . . . . .	161
B.10 Survey # 4 Results: Leap Usability . . . . .	162
B.11 Survey # 4 Results: Perceptions of Leap . . . . .	163
B.12 Survey # 4 Results: Lessons Learned . . . . .	163

# List of Figures

<u>Figure</u>	<u>Page</u>
1.1 Eras of Software Development Improvement . . . . .	2
1.2 Leap Toolkit Controller. This is the main controller for the Leap toolkit. The developer may start data recording tools or start tools to modify their definitions. . . . .	4
1.3 Hee Project Viewer. This tool allows the developer to define, plan and analyze a single project. Cam has filled out the name, description and start date for the Multi-User Calendar project. He has also decided to use his Development process for this project. . . . .	5
1.4 Io time recording tool. Io allows the developer to easily record the amount of time they spend working on a task. They may also account for any interruptions by recording interrupt time. In this Figure Cam has worked for 14 minutes on the design of the Multi-User Calendar. . . . .	6
1.5 Project Comparison Tool showing the summary for Cam's Java size data. On average he has 18 LOC per method. . . . .	7
1.6 Time Estimation Tool. The time estimation tool shows Cam's historical data. Cam has chosen Linear Regression for the trend lines and Lines of code as the size measures for time estimation. The planned size 3024 is taken from Hee. Based upon this data the project should take from 1656 to 2290 minutes. . . . .	8
2.1 PSP levels . . . . .	15
2.2 PSP0.1 phases . . . . .	16
2.3 PSP Time Estimation procedure . . . . .	18
2.4 PSP2.0 phases . . . . .	19
2.5 PSP3.0 phases . . . . .	19
2.6 Time Recording Log after Cam has started planning for the Multi-user extension project. . . . .	21
2.7 PSP2 Project Summary form after Cam has filled in the header information for the Multi-user extension project. . . . .	22
2.8 Conceptual Design form after Cam has developed a conceptual design for the multi-user project. . . . .	23
2.9 Size Estimation form after Cam has developed a conceptual design for the multi-user project. . . . .	25
2.10 PSP2 Project Summary form after Cam has filled in the planned size information for the Multi-user extension project. . . . .	26

2.11	PSP2 Project Summary form after Cam has filled in the planned time information for the Multi-user extension project. . . . .	27
2.12	PSP2 Task Planning Template after Cam has filled in the planned task information for the Multi-user extension project. . . . .	28
2.13	PSP2 Schedule Planning Template after Cam has filled in the planned schedule information for the Multi-user extension project. . . . .	29
2.14	PSP2 Project Plan Summary after Cam has finished planning the defect portion of the Multi-user Calendar extension project. . . . .	30
2.15	Time Recording Log after Cam has finished planning the Multi-user extension project.	31
2.16	Time Recording Log after Cam goes to the staff meeting. . . . .	32
2.17	Cam's Design Review Checklist for Java. . . . .	32
2.18	Cam's Defect Recording Log showing one design defect found in the design review. It took him one minute to fix the defect. . . . .	33
2.19	Time Recording Log after Cam has finished his design review and started coding. .	33
2.20	Cam's Defect Recording Log showing the design defect found in the design review and the design defect found during coding. It took him three minutes to fix the second defect. . . . .	34
2.21	Cam's Test Report Template showing the first test that tests all the constructors. The test eventually passed. . . . .	35
2.22	Cam's Filled in PSP2 Project Plan Summary form after counting all the recorded defects. . . . .	36
2.23	Cam's Filled in PSP2 Project Plan Summary form after counting the program size with LOCC. . . . .	37
2.24	Cam's Filled in PSP2 Project Plan Summary form after adding up all the time entries from the Time Recording Log. . . . .	38
2.25	Generic Review Process . . . . .	48
2.26	Spectrum of Formal Technical Reviews . . . . .	49
3.1	A short section of a Leap data file. Notice the HTML tables. Each row in the table is a data entry. . . . .	65
3.2	Leap File IO architecture. . . . .	66
3.3	Leap data file loading and saving example. . . . .	67
3.4	Model-View-Controller architecture in the Leap toolkit. . . . .	68
3.5	Leap package architecture . . . . .	68
3.6	Defect analysis, number of defects and the total amount of time it took to fix them per defect type. . . . .	70
3.7	Defect analysis, number of defects injected per phase. . . . .	70
3.8	The Leap toolkit extension dialog. The Leap toolkit found only one extension in its extension directory. . . . .	71
3.9	Leap Toolkit Controller. This is the main controller for the Leap toolkit. . . . .	72
3.10	Hee, the project editor. Cam has defined the Multi-User Calendar extension project and chosen his Development process. . . . .	73
3.11	Io time recording tool. Cam has just started recording time for the Design phase of the Multi-User Calendar extension project. . . . .	73
3.12	Io time recording tool. Cam has just interrupted recording time to answer the phone.	73

3.13	Io time recording tool. Cam has just started his planning phase. . . . .	74
3.14	Project Comparison tool. Cam is comparing all his Java development projects to see his average LOC/Method. . . . .	74
3.15	Hee Size tab. Cam has entered in his planned size for the Multi-user Calendar project.	75
3.16	Time Estimation Tool. Cam has chosen to estimate based upon his planned sizes, linear regression model, and methods. . . . .	76
3.17	Hee Time Tab. Cam has added six different time estimations based upon different trend lines and size categories. . . . .	77
3.18	Project Comparison Time Tab. Cam is comparing all his Java development projects to see how his time is distributed between the phases. . . . .	78
3.19	Hee Time Tab. Cam has filled in his planned time for the project. . . . .	79
3.20	Hee Combined Tab. The Leap toolkit has calculated Cam's expected productivity for the Multi-User Calendar project. . . . .	80
3.21	Loading a Leap data file from an URL. . . . .	81
3.22	Defect Recording Tool Mano. Shows a design defect in the Multi-User Calendar project. The defect is a bad parameter defect where a parameter is missing from a constructor. . . . .	81
3.23	Email Review Data window. The reviewer is sending their defect data to Cam. Only the defect data will be attached to the email message. . . . .	82
3.24	Defect Recording Tool, Mano, showing the design defect found during coding. Cam has just started to fix the defect. The timer has just started. . . . .	83
3.25	Cam's Code Review Checklist. . . . .	84
3.26	Hee project summary showing the Multi-User Project size estimates and actuals. Cam was close in his size estimation. . . . .	85
3.27	Hee project summary showing the Multi-User Project time estimates and actuals. Cam was very close in his time estimation. He underestimated the amount of time by less than 10%. . . . .	86
3.28	Hee project summary showing the Multi-User Project defects removed. Cam removed 59% of the recorded defects before the compile phase. . . . .	87
3.29	Hee project summary showing the Cam's productivity. Cam worked much faster than he expected. . . . .	88
3.30	Time Planning Accuracy chart. Cam's planning accuracy has been oscillating between 50% and 150%. . . . .	89
3.31	Defect analysis, number of defects and the total amount of time it took to fix them.	90
3.32	Time Estimation Tool performs all the calculations for Cam. . . . .	92
3.33	Time Recording Log after Cam goes to the staff meeting. . . . .	93
3.34	Time Recording Log after Cam goes to the staff meeting. . . . .	95
3.35	Naia, the time recording tool, with the correct duration . . . . .	96
3.36	Default values inserted into Naia: Project, Phase, and Start date. . . . .	97
3.37	Pop-up menu for the currently defined projects. . . . .	98
3.38	Time table (Naia) with a time entry that is in error. . . . .	99
3.39	The Leap toolkit groups all undefined values into the <i>&lt;Undefined&gt;</i> category. . . .	101
3.40	Moa, Project Comparison tool project selection screen. . . . .	102
5.1	LeapDet. Simple defect entry tool. . . . .	114

5.2	LeapDat. Simple defect analysis tool. . . . .	115
5.3	Naia time entry/editing tool. . . . .	118
5.4	Io time recording tool. . . . .	119
5.5	Students' percentage of time entries that are a multiple of 5. . . . .	120
5.6	Students' percentage of start and stop time entries that the same or different by 1 minute. . . . .	121
5.7	A simple Box and Whisker plot. . . . .	123
5.8	Size estimation accuracy (LOC). The class' average accuracy leveled off. The class tended to underestimate how large the projects would be. . . . .	123
5.9	Size estimation accuracy (Methods). The class' average accuracy increases over time. The class tended to underestimate how large the projects would be. . . . .	124
5.10	Time estimation accuracy. The class' average accuracy increases over time. The class tended to underestimate how long the projects would take. . . . .	125
5.11	Class productivity in LOC/hr. The class average increases slightly over the last 3 projects. . . . .	133
5.12	Class productivity in Methods/hr. The class average stays roughly the same over the last 3 projects. . . . .	133
5.13	Average LOC/Method. The class average increases over time. . . . .	134
A.1	Leap survey #1 page 1 . . . . .	147
A.2	Leap survey #1 page 2 . . . . .	148
A.3	Leap survey #2 page 1 . . . . .	149
A.4	Leap survey #2 page 2 . . . . .	150
A.5	Leap survey #2 page 3 . . . . .	151
A.6	Leap survey #3 page 1 . . . . .	152
A.7	Leap survey #3 page 2 . . . . .	153
A.8	Leap survey #4 page 1 . . . . .	154
A.9	Leap survey #4 page 2 . . . . .	155
A.10	Leap survey #4 page 3 . . . . .	156

# Chapter 1

## Introduction

*At the start, when we know much about the problem and nothing about the solution, the solution is very abstract.* – Robert H. Dunn

Virtually every software developer wishes they got home from work sooner and spent less of their weekend online. Software developers work very hard and very long hours, yet software is often delivered late, over budget, and full of defects. Over forty years of software development experience has not succeeded in conquering this problem. How can software developers gain more control over their software development and produce high quality software efficiently? The Leap toolkit contributes a new weapon to the battle for high quality software delivered on time and within budget. Before I discuss Leap I will give a short background of the software development problem.

### 1.1 Why is Quality Software Development Important?

Software is controlling more safety critical tasks and important functions[1]. Yet software errors occur sometimes with horrible costs. Between 1985 and 1987 the Therac-25 radiation therapy machine killed two people and seriously injured four others by delivering massive radiation overdoses. Investigations found that many issues were to blame including faulty software[27].

In March 1995, the Denver International Airport opened over 16 months late and over 100 million dollars over budget. One of the primary reasons for the delay and overrun was the presence of major bugs in the baggage handling control software[11].

Another problem with current software development is productivity. The insatiable demand for more software has out-paced our ability to produce software. Software productivity has not kept pace with hardware cost/performance ratios. The most optimistic rate at which programmer productivity is increasing is 5% per year, while there is greater than a ten fold increase in the



demand for software each decade[6]. The government of the United States of America faces this problem. In 1996, Computerworld reported that delays in overhauling the federal tax computer systems cost the U.S. Treasury as much as \$50 billion per year[11]. The problem of overhauling of the tax computers is not purely a software issue but the software system is a large part of the problem.

## 1.2 Traditional Solutions

Software developers and managers have addressed software quality and development issues since the beginning of the computer age. Developer and managers have continuously augmented the development methods they use. One way to envision these efforts is a four overlapping, complementary “eras”: hope-based, product-based, organization-based, and individual-based. Figure 1.1 shows a time line of software development improvement and the different eras.

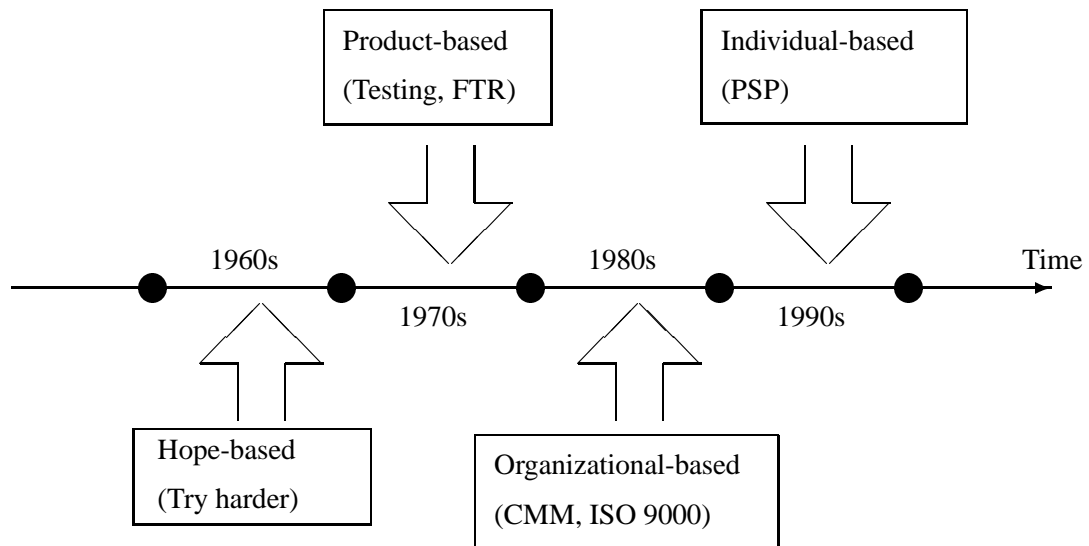


Figure 1.1. Eras of Software Development Improvement

In the 1960's, most software development improvement efforts were just focused on doing better. The problems of software development tended to be small since computers were very small as compared to today's computers. "Just trying harder" seemed like a reasonable solution.

In the 1970's, people realized while the "try-harder" method did help improve software development, it was not enough. Computers and software programs were getting more complex and

new methods were needed. Developers and researchers started looking at the work products. Many developers and computer scientists began to advocate structured testing to help improve the quality of software. In 1976, Fagan reported a group-based review method called on Software Inspection[8] as a means to efficiently improve the quality of the work product.

In the late 1980's, the focus shifted from the work products to the organizations that produced the work products. The Software Engineering Institute (SEI) introduced the Capability Maturity Model[30] and ISO 9000[19] became wide spread. These methods attempted to improve software quality by recommending organizational structures and procedures.

In the late 1990's, yet another focus has emerged: individual software engineering processes. In 1995, Humphrey introduced the Personal Software Process[17], a software development process and improvement process for individual software developers. The PSP is a manual process where the developer collects data about the amount of time they spend, the size of the work product, and the defects they make while developing software. By analyzing the data at the end of the project, the developer gains insights into their development process. These insights help the developer increase productivity and work product quality. In other words, high quality software developers produce high quality software.

Several studies have shown that the PSP appears to help improve software development [9, 14, 26, 33], but is not the complete solution to the software development issue.

Another study of the data collected in the PSP by Anne Disney[4, 5] showed that there are significant issues of data quality in the manual PSP. Disney divide the types of data error into seven categories. The combination of collection and analysis errors calls into question the accuracy of manual PSP results.

After using the PSP for two years in our research group, the Collaborative Software Development Laboratory (CSDL), we decided to try to build upon the PSP's strong foundation and incorporate features of formal technical review to produce a more effective software developer improvement tool.

### **1.3 LEAP: Giving developers more control and insight**

LEAP is a design philosophy intended to produce effective tools that allow developers to gain valuable insight into their own software development. Using the LEAP design philosophy I developed the Leap toolkit, a Java application that supports technical skill acquisition. Throughout the development of the Leap toolkit, I used feedback from actual users of the toolkit to guide changes

and improvements. This style of development is similar to action research[13]. The users and I were active participants trying to improve their experiences with individual software engineering improvement.

### **An example of the Leap toolkit**

I will introduce the Leap toolkit by using a hypothetical software developer, Cam, and his manager, Philip. Cam and Philip work for a small world class Java software development company. Cam has been using the Leap toolkit for about a year to keep track of his Java programming projects. He has a small database of over 30 Java projects.

### **Starting a new project**

Philip calls Cam into his office to discuss Cam's next project. The next project is an extension to their single user calendar tool that allows multiple users to use the same calendar while keeping some events private. Philip gives Cam the requirements for the new extension and ask him how long the project will take. Cam tells Philip that he needs to do some design work before he can give an accurate estimate. Cam goes back to his cubicle and starts the Leap toolkit as shown in Figure 1.2.

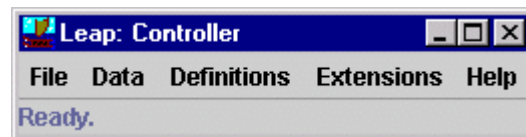


Figure 1.2. Leap Toolkit Controller. This is the main controller for the Leap toolkit. The developer may start data recording tools or start tools to modify their definitions.

### **Defining the new project**

To start the new project Cam opens the Projects (Ilio) tool. To create a new project Cam starts the project editing tool Hee on the first blank line. He types in the name of the new project "Multi-User Calendar" and a brief description of the project. Then he selects the start date for the project and chooses the PhaseSet that he plans on using. The PhaseSet is a set of phases that describe his development process. Cam evolved his current PhaseSet after experimenting with different development processes. Figure 1.3 shows the Hee project viewer.

The screenshot shows a window titled "Hee project viewer" with a standard Windows-style title bar. Below the title bar, it says "Summary data for project: 03/23/2000". There are five tabs: "General", "GQM", "Size", "Time", and "Defects", with "General" being the active tab. The "General" tab contains several fields: "Name:" with the value "Multi-User Calendar"; "Description:" with the text "This is an extension to our single user calendar to support multiple users sharing the data and having private data."; "Start Date:" with a date/time value of "8:15:59 AM 13-Mar-00"; "End Date:" which is empty; "FileList:" which is empty; and "PhaseSet:" with a dropdown menu showing "Development" and a "Change PhaseSet" button. At the bottom of the window, there are two buttons: "Condense Raw Data" and "Close".

General	GQM	Size	Time	Defects	Combined
<b>Name:</b> Multi-User Calendar					
<b>Description:</b> This is an extension to our single user calendar to support multiple users sharing the data and having private data.					
<b>Start Date:</b> 8:15:59 AM 13-Mar-00					
<b>End Date:</b> 					
<b>FileList:</b> 					
<b>PhaseSet:</b> Development <span>Change PhaseSet</span>					
<span>Condense Raw Data</span> <span>Close</span>					

Figure 1.3. Hee Project Viewer. This tool allows the developer to define, plan and analyze a single project. Cam has filled out the name, description and start date for the Multi-User Calendar project. He has also decided to use his Development process for this project.

## Developing an initial design

Cam then starts the Io timer tool to record the time he spends designing the new project. Figure 1.4 shows the Io timer after he started recording time for the design phase.

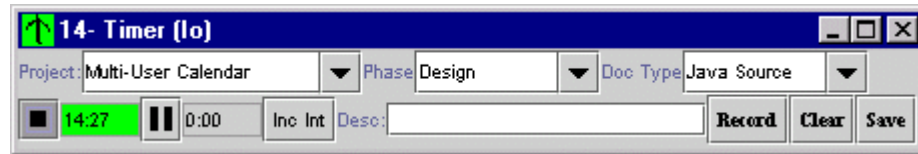


Figure 1.4. Io time recording tool. Io allows the developer to easily record the amount of time they spend working on a task. They may also account for any interruptions by recording interrupt time. In this Figure Cam has worked for 14 minutes on the design of the Multi-User Calendar.

Cam works on the design for the new project and when he finishes his design he clicks the stop button on Io and then records his time in the Leap toolkit. Cam's initial design calls for 30 classes and 168 methods.

## Project Planning

**Size Planning** Cam then opens the Project Comparisons tool to find out his average lines of code per method. Figure 1.5 shows the Leap toolkit's Project Comparison tool. His average is about 18 lines of code per method. Using this figure and the fact that his design indicates that 168 methods will be required, he calculates that the whole project will be 3024 lines of code. He reopens the Hee project viewer for the Multi-User calendar project and enters in his planned sizes.

**Estimating effort** He then goes to the time window in Hee and starts the time estimation tool. Cam choose to use linear regression model for the trend lines and lines of code for the size grain size then presses the estimate button. Figure 1.6 shows the time estimation tool with the estimate. This tool displays historical data on the size and time of previous projects. It also allows the user to generate various predictions for the time required for a current project given a size estimate and using the historical data as a basis. Based upon his historical data this project will take about 1973 minutes. Cam tries some of the other combinations like methods and linear regression model to give him a range of estimates. Based upon all these estimates, Cam estimates that it will take him anywhere from 28 1/3 hours to 38 2/3 hours of direct work to complete the project. Cam enters in his time estimate into Hee and then sets up a meeting with Philip. From Cam's historical time data he knows that he only works an average of 6 direct hours of work per day. Since Cam is working on

The screenshot shows a 'Project Comparisons' dialog box with a blue title bar and standard window controls. It features four tabs: 'Size', 'Time', 'Defects', and 'Combined', with 'Size' currently selected. The dialog contains a table with three columns: 'Estimation Analyzer', 'Total', and 'JavaSize'. The table lists various metrics for Java size, including LOC, Method, Class, Package, and LOC/Method, LOC/Class, and LOC/Package. An 'Ok' button is located at the bottom center of the dialog.

	Estimation Analyzer	Total
	JavaSize	JavaSize
LOC	1,069 LOC	23,928 LOC
Method	35 Method	1,305 Method
Class	1 Class	227 Class
Package	0 Package	58 Package
LOC/Method	30.543 LOC/Method	18.336 LOC/Method
LOC/Class	1,069.000 LOC/Class	105.410 LOC/Class
LOC/Package	∞ LOC/Package	412.552 LOC/Package

Figure 1.5. Project Comparison Tool showing the summary for Cam's Java size data. On average he has 18 LOC per method.

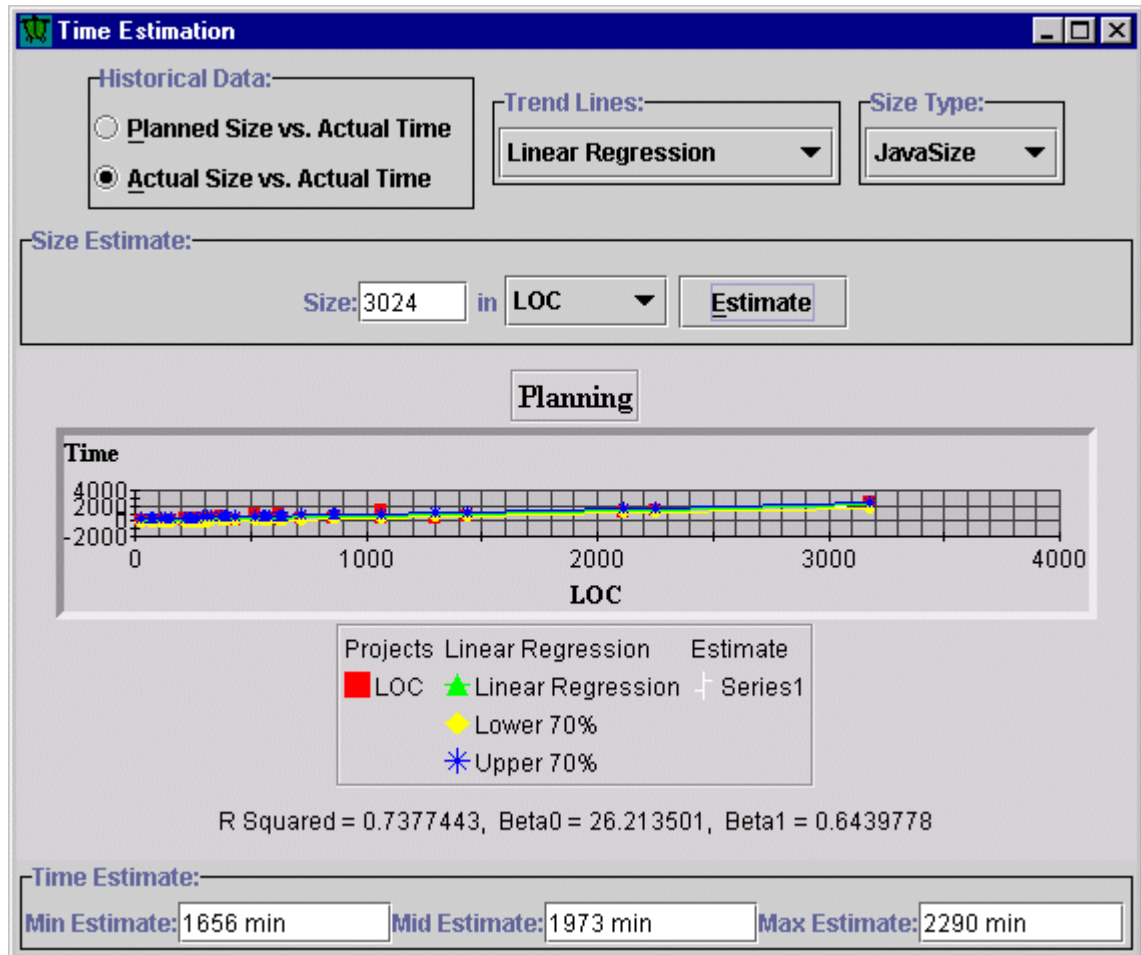


Figure 1.6. Time Estimation Tool. The time estimation tool shows Cam's historical data. Cam has chosen Linear Regression for the trend lines and Lines of code as the size measures for time estimation. The planned size 3024 is taken from Hee. Based upon this data the project should take from 1656 to 2290 minutes.

three project concurrently, he can only dedicate 2 hours a day of work on the Multi-User Calendar project, so the  $28 \frac{1}{3}$  to  $38 \frac{2}{3}$  direct hours will take from 14 days to 20 days to complete. If Cam is able to use all 6 hours per day on the project, it should take between 5 and 7 days.

## **Negotiation**

At the meeting with Philip, Cam tells Philip that the project will take him between three and four weeks. Philip wants the project done in one week. Cam replies that this is only possible if he can stop working on his other projects and focus solely on the Multi-User Calendar project. Philip agrees that Cam may drop his other projects until the calendar is finished.

In this example, the Leap toolkit allowed Cam to predict how long the Multi-user calendar project will take to develop, and to negotiate effectively with his manager to obtain the resources required to complete the project in a timely manner. Cam's prediction is based upon his historical productivity for Java programs and his estimated size of the project. The Leap toolkit facilitated the collection and analysis of all the data required for making the prediction. Manual methods, such as the PSP, also support data collection and analysis, but with a much higher overhead to the user.

## **1.4 Thesis Statement**

The Leap toolkit provides a novel tool that allows developers and researchers to collect and analyze software engineering data. It addresses many problems regarding data quality found in prior research, and provides novel support for empirically-based personal software engineering.

## **1.5 Investigating Individual Software Development**

I used the Leap toolkit to investigate two important issues in software development, collection error and project estimation.

First, The Leap toolkit addresses the issue of collection stage errors by reducing the user's overhead of data collection. To investigate data collection errors, I conducted a case study of 16 graduate students in an advanced software engineering course at the University of Hawaii. During the case study I conducted four surveys, interviewed 11 students, and analyzed the software development data collected by the students. The case study indicates that reducing the overhead of data collection may reduce some collection errors.



Second, the Leap toolkit provides data analyses that are not practical with a manual method. I conducted an empirical experiment to determine if any particular time estimation technique is more accurate. Using the Leap toolkit, I compared 13 different time estimation techniques to determine if any of them are more accurate in predicting the effort for a new project. I found that for two students, they were much better at prediction than any of the other methods. For a third student, one of the estimation methods was more accurate. If this student had known this, they could have made more accurate predictions. Furthermore, combining the data sets, I found that across the whole class the students were significantly better estimators than any of the other time estimation techniques.

## 1.6 Contributions

This research provides the following contributions to the software engineering community: the LEAP design philosophy, the Leap toolkit, and the results of the case study. One contribution of this research is the LEAP design philosophy provides a set of guidelines that address some of the issues with traditional software improvement efforts.

Another major contribution is the Leap toolkit. I have made the Leap toolkit freely available on the Internet. Software developers may download the Leap toolkit and use it in their own work. The Leap toolkit has been available for over one year and many developers have downloaded it. The Leap toolkit also provides a novel tool for software developer education, as has been demonstrated in several graduate and undergraduate software engineering classes. Instructors can gain insight into how the students are actually spending their time and provide more detailed help.

Another contribution are the results of the case study. The results indicate that the Leap toolkit reduces the overhead of data collection and that high overhead in data collection correlates to rounding error. The Leap toolkit's sophisticated size representation, tool support for counting size, and different estimation models allowed me to compare 13 different time estimation methods to determine which method was most accurate for each student and the class. I determined that, for the class, the best estimation technique was using an exponential regression on the actual number of methods. This may indicate a fundamental relationship in software engineering that the relationship between size and time is not linear.

## **1.7 Organization of the Dissertation**

This dissertation is organized as follows. Chapter 2 relates the current research to the broader context of existing work. Chapter 3 depicts the main design features and planned benefits of LEAP. Chapter 4 outlines the methods I used to investigate collection errors and time estimation. Chapter 5 presents the results of the investigations. Finally, Chapter 6 presents some concluding remarks on Project LEAP and some future directions for this research.

# Chapter 2

## Related Work

*If you don't know what you are doing, it is hard to improve it. – Watts Humphrey*

Project LEAP and the Leap toolkit are a result of our experiences using the Personal Software Process (PSP) for over three years, our experiences with Formal Technical Review (FTR) and our attempts to improve the quality of software development.

The PSP provides the fundamental ideas for personal developer improvement used in Project LEAP. The PSP teaches developers how to collect data on their own development processes. At the end of a project the developer analyzes the data that they collected and look for areas that need improvement and areas that show strength. The PSP also allows the developer to plan their projects based upon their past experiences. I learned how a developer can improve their development process by collecting and analyzing the right kinds of data. The PSP provides the basic ideas of what data to collect and some of the data analysis that are possible and helpful in process improvement. The PSP shows how personal process improvement is possible.

FTR is a powerful method for efficiently improving the quality of work-products. A group of technical people gather to review a work-product or process. Each reviewer reviews the work-product and looks for defects or issues. These issues are collected together and the author of the work-product fixes the defects. The different perspectives that others bring to the review often find defects that would not be found. Reviews are a powerful mechanism for gaining insight into the types of defect a developer makes, but does not find on their own. Project LEAP attempts to combine the PSP and FTR to provide powerful insights for the developer.

Both the PSP and FTR require practitioners to collect detailed data about the development process and work-products. This data is very personal and can be misused. All of this data collection can lead to Measurement dysfunction. Measurement dysfunction is the situation where

the measures look good, but the actual behavior is not good. The PSP and FTR practices require that measurements are taken to help improve the processes. Any tool that supports the PSP and FTR must address the issue of measurement dysfunction since the measures are used to improve the processes. If the measurements do not reflect the actual behavior then the improvement effort is misguided.

This chapter briefly discusses the PSP, some of the different tools developed to support the PSP, and some evaluation of the PSP. Next, it discusses the software quality assurance process called Formal Technical Review. The chapter concludes with a discussion of measurement dysfunction and some of the data quality issues found in the PSP and FTR.

## 2.1 Personal Software Process

The Personal Software Process[17] is a self-improvement process for software developers. In his book “A Discipline for Software Engineering”, Watts Humphrey teaches software developers how to become their own software development coaches. Sports coaches observe the performance of their players, evaluate their performance, then make suggestions for improvement. This is the classic *observe, evaluate, modify* cycle for improvement.

In the PSP, the developer observes their performance by recording how they develop software. They record, on paper forms, the amount of time each phase takes, all the defects they make and the size of the work done. After each project they evaluate how they performed by conducting standard analyses on the data they collected. Based upon these project postmortems, the developer should gain insight into their development process, and then modify it in an attempt to improve it. After modifying their development process, the developer starts the cycle again. The cycle consists of the following steps.

- *Observe*: While they are developing their next project they record how long it takes them, the number of defect they make during development, and the size of the work-products they develop.
- *Evaluate*: During the Postmortem phase of the project, they assess whether their modifications actually helped their software development. The developer may not be able to assess the results of their modification after just one project. For example, developer Cam adds a design review phase to his development process and records one project using his new development process. During postmortem, he notices that the total number of defect detected and removed

from the project was higher than his historical average. Cam cannot jump to conclusions and say the adding design reviews improved his development because he caught more defects. The project may have been in a new area and he was not familiar with the domain and made more simple defects not detected in the design review. Maybe this one project was an exceptional project that cannot be repeated. Cam should try his modified process for a few projects and see if the increased defect detection continues.

- *Modify*: The developer then initiates changes to their development process based upon their analysis and starts the whole cycle over again. A simple example of changing the development process is if Cam notices, based upon his evaluation of his defect data, that he tends to make logic errors in his `if` statements. Cam decides to add a checklist item in his code review that specifically checks on the logic of all `if` statements. Cam can observe the number of times this new checklist item finds a defect in his code in future projects. If the checklist item reduces the number of logic defects found during testing then Cam has improved his development process.

Software developers using the PSP become their own software development coaches.

### **2.1.1 Purpose of the PSP**

There are many possible goals in software development improvement. The PSP focuses on just two:

- Developers using the PSP should produce high-quality software efficiently, and
- Developers should improve their ability to estimate the amount of effort required to produce the software.

These two goals drive the whole PSP. The data collection and analyses are focused on improving the developer's software development and estimation skills.

### **2.1.2 Learning the PSP**

To teach developers how to use the PSP, Humphrey defines seven PSP processes (0, 0.1, 1.0, 1.1, 2.0, 2.1, 3.0). Each process has detailed scripts telling the user exactly how to perform the process. Figure 2.1 shows the seven levels. Exercises at the end of each chapter in "A Discipline for Software Engineering" ask the reader to use the knowledge from the chapter to improve their

development skills. The chapters introduce powerful development techniques, including: design and code reviews, size and time estimation methods, and design templates. These techniques help the developer produce high quality products efficiently. As developers go through the book they develop 10 small software projects using the different PSP levels.

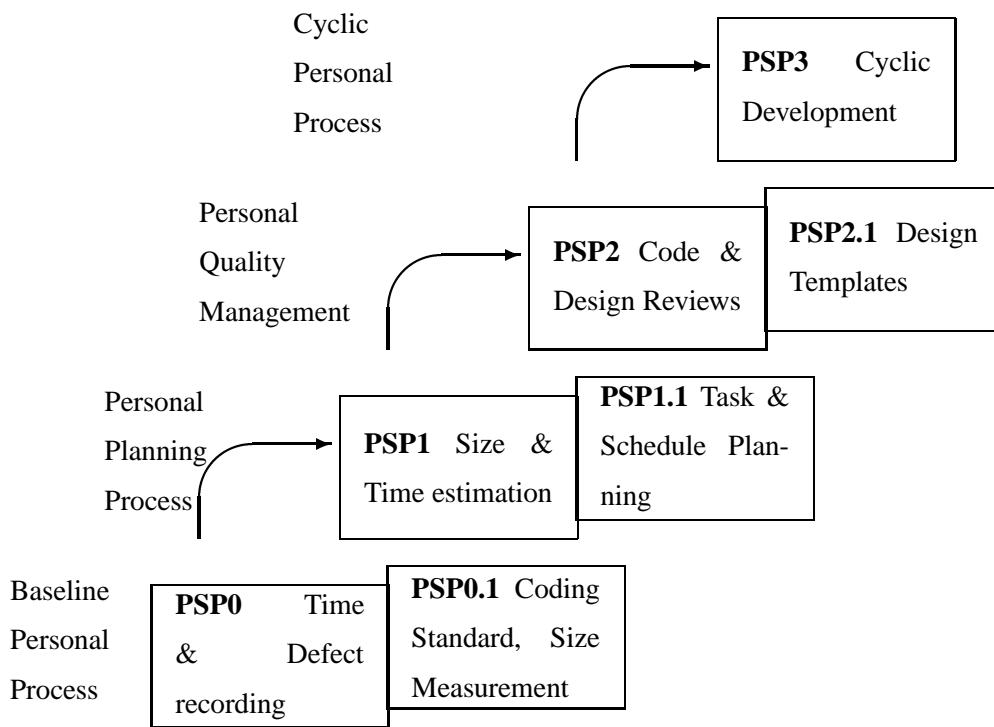


Figure 2.1. PSP levels

### 2.1.2.1 PSP0: The Baseline Process

The baseline processes PSP0 and PSP0.1 introduce the concepts of data collection and size measurement to the developer. The purpose of these processes is to give the developer a basis for their improvement. By recording the time they spend, the defects they make and the size of

their work, the developer learns exactly how they develop software. To facilitate the collection and analysis of all this data the developer learns to use the Time Recording Log, Defect Recording Log and Postmortem forms to record and analyze time, size and defect data. The two logs help the developer record all the that necessary for later analysis. The Project Plan Summary form helps the developer summarize their data for the project. To assist in analysis and support good development practice, the developer must follow the following development phases: planning, design, code, compile, test and postmortem phases. Figure 2.2 shows the order of the phases.

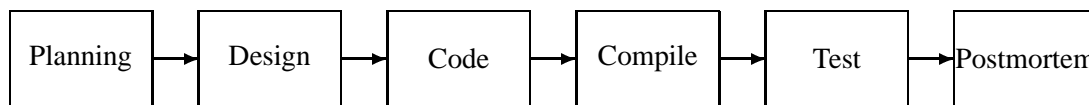


Figure 2.2. PSP0.1 phases

To help the developer fulfill the requirements of each phase, PSP0 has four scripts describing exactly what the developer should do for each phase. In the planning stage they make their “best guess” as to how long the project will take. This introduces the concept of making a plan before starting the project. This concept is expanded upon in the next PSP level. In the postmortem phase the developer fills out the Project Summary form and reflects on their development process and the project that they just finished.

#### **2.1.2.2 PSP1: The Personal Planning Process**

In PSP1 the developer adds a detailed Planning phase, with size and time estimation, to their development. In the planning phase they make explicit, documented plans for their work. The PSP planning method uses proxy based estimation (PROBE). In PROBE the developer uses a “proxy” to estimate the size of in LOC. A proxy is a substitute or stand-in that the developer can easily estimate and visualize. The proxy used in PSP is the method or function. Most developers cannot directly estimate the number of LOC in a program, but they can give a good estimate of the number of methods it would take. Based upon historical size data about previous work-products, the developer calculates five typical size categories for their methods. When they start a new project they develop a conceptual design and determine the size category for each method. Based upon the

number of methods in each category, the developer can determine the total size of the project. This is the initial planned size of the project. The PROBE method uses a linear regression model of planned size to actual size to compensate for under or over planning. The results of the linear regression model is the planned size for the project. See Section 2.1.3.1 for more details on this process. Now that the developer has an estimated size for the project they use the PSP time estimation process. This process is shown in figure 2.3.

In cases where a significant ( $r^2 > 0.50$ ) correlation exists between planned size and actual time, the developer uses a linear regression model based upon these historical values to predict project time from planned size. If the correlation between planned size and actual time is not significant, the developer checks the correlation between actual size and actual time. If their actual size and time data is not highly correlated then they use their actual size and actual time averages. During the postmortem phase the developer compares their plan to their actual performance. The developer calculates their CPI, this ratio is their planned time over their actual time. It indicated whether they are under or over planning their projects. This ratio can be used to adjust their future plans. PSP 1.1 adds the concepts of task and schedule planning. This allows the developer to better estimate and schedule their projects. PSP level 1 addresses the second goal of the PSP, the next level addresses the first goal of efficiently produced high-quality software.

### **2.1.2.3 PSP2: Personal Quality Management**

PSP2 introduces quality control measures by adding two reviews to the process. The developer reviews their own design before they start coding and they review their code before they start compiling. Review is a powerful quality improvement process. Many studies have shown that removing defects early in the development process is cheaper than removing those defects later in the development process. The two reviews should catch defects earlier and reduce the cost of fixing defects. PSP2.1 addresses the design process by introducing design templates, logic and state diagrams these tools should help the developer produce more correct programs with less overall effort. The purpose of PSP2 is to provide the developer with tools to efficiently improve the quality of their work products. As a part of the project postmortem the developer calculates the yield for their defect removal efforts. The yield for the project is the number of defects removed before the first compile divided by the total number of defects removed in the whole project. The developer may gain insight into good quality practices by calculating their yield and reflecting on what made this project better or worse than their other projects.



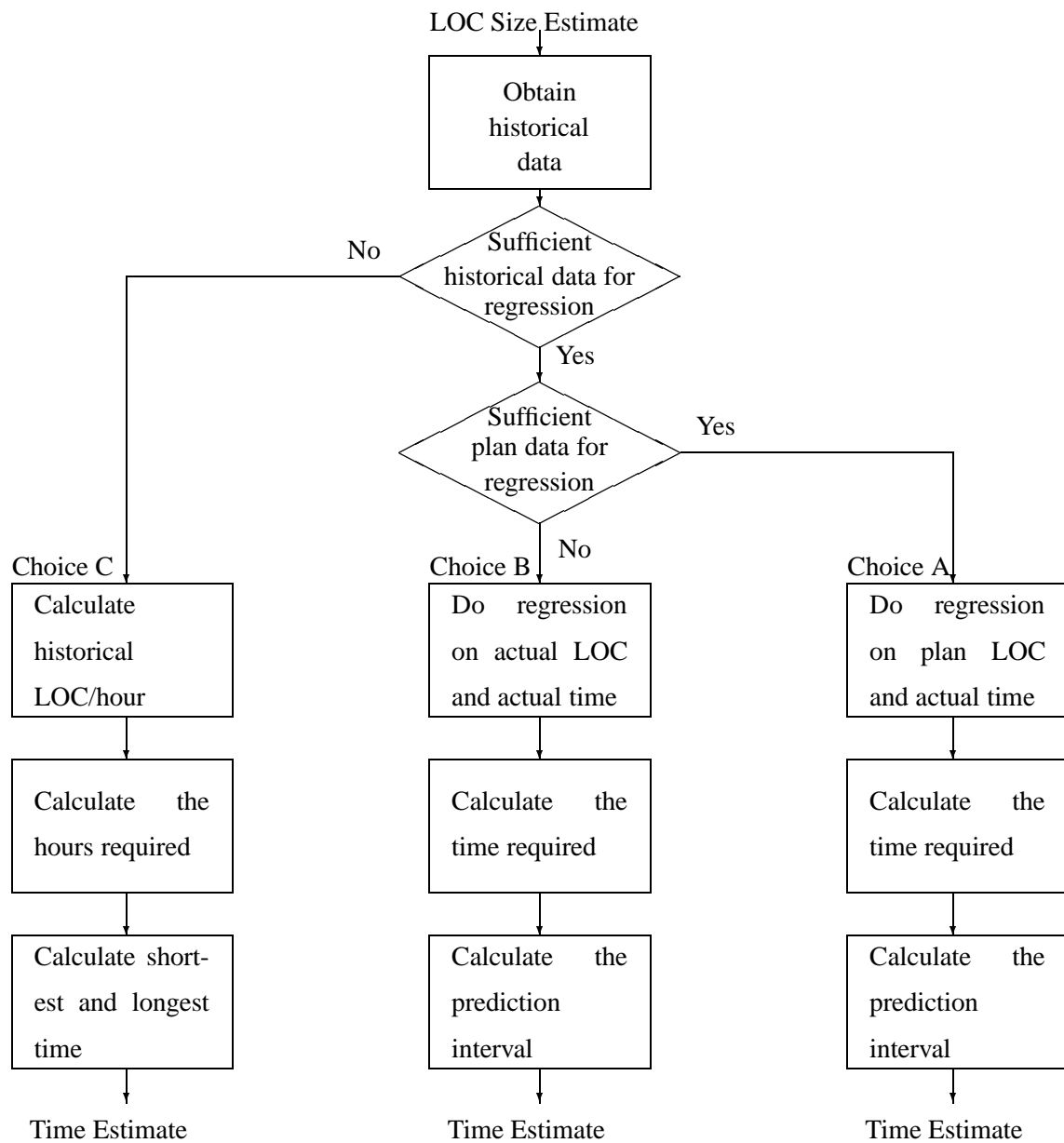


Figure 2.3. PSP Time Estimation procedure

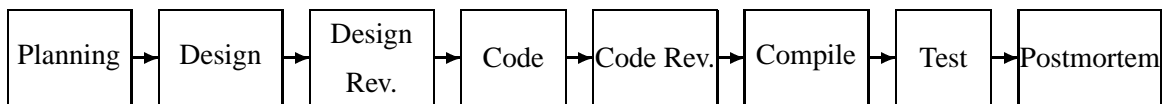


Figure 2.4. PSP2.0 phases

Up to this point the PSP has focused on small projects. To address larger projects the PSP introduces the Cyclic Personal Process.

#### 2.1.2.4 PSP3: Cyclic Personal Process

PSP3 changes the overall development process from a strict linear waterfall model to a cyclic spiral model. PSP3 allows the developer to subdivide a larger program into smaller pieces that are developed using PSP2. Figure 2.5 shows the new development process. The whole program is built up of enhancements on the previously completed increments. This builds up a high quality final product as long as each increment is of high quality. The purpose of PSP3 is to expand the PSP to larger projects.

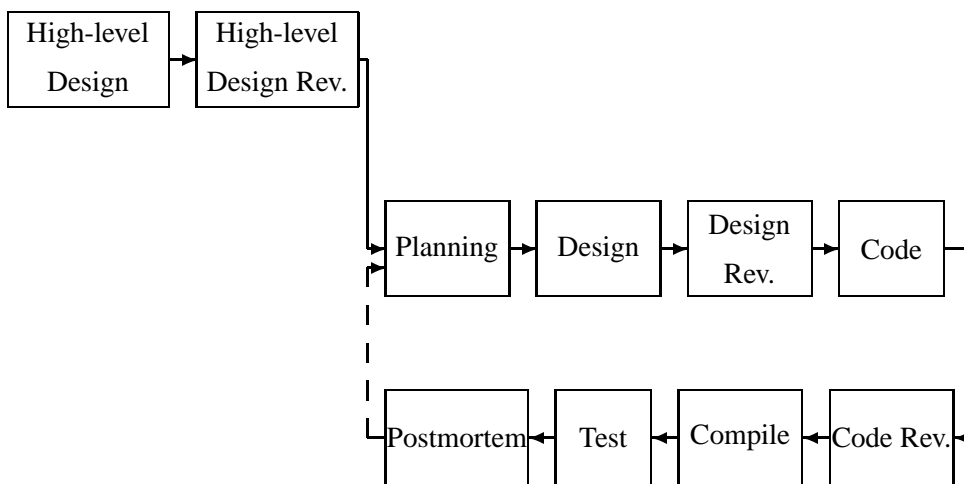


Figure 2.5. PSP3.0 phases

### 2.1.3 Using the PSP

There is a distinction between the PSP process and the PSP curriculum. Humphrey says that the PSP should be modified by the user to support their own goals and situation. However, the developer should not modify the process of learning the PSP — they must go through all the stages to learn how to properly use the PSP before they modify the process.

Modifying the PSP requires the developer to adhere to the PSP's forms. The forms in the PSP are very tightly tied to the PSP process. Small changes to the process like renaming the phases or other cosmetic changes are not difficult. Larger changes like changing the size measurement or estimation technique are very difficult and may affect many forms and scripts. The analyses that the PSP calls for may be invalidated with even minor changes to the PSP process. Modifying the PSP may also invalidate the developer's historical data that has been carefully collected, analyzed and maintained.

#### 2.1.3.1 An Example use of the PSP

The last sections described the PSP process at an abstract level. To make the discussion more concrete, this section describes one scenario of use. This example has two main characters, Cam, the software developer, and Philip, Cam's manager. Cam and Philip work in a small software development company. They have built a calendar tool for the Palm<sup>TM</sup> Pilot hand held computer.

**Starting a new Project** Cam is called into Philip's office to get the requirements for the next project. "We need to extend our existing Palm calendar application by adding multiple user support." Cam and Philip go over the requirements for the new Multi-user features. By the end of the meeting Cam understands the requirements for the new features. Philip asks Cam how long the project will take. Cam replies that he needs to analyze the requirements a bit more and will give Philip an answer in a day.

**Planning** Cam returns to his cubicle and decides to use PSP level 2.0 for this new project. To start the project Cam gathers all the necessary inputs to the project.

- Problem description. Cam uses the requirements document from Philip and the notes he took during the meeting.
- PSP2 Project Plan Summary form.

- Size Estimating Template.
- Historical estimated and actual size and time data.
- Time and Defect Recording Logs.
- Defect Type Standard.

Cam pulls out a copies of the PSP2 forms from his filing cabinet. He also digs out all his old Project Summary Forms to get the Historical estimated and actual size and time data.

Cam can now start the new project, he fills in the header information on the Time Recording Log (TRL) and enters the start time. Figure 2.6 shows the TRL.

**Table C16a Time Recording Log**

Student		Cam Moore				Date	3/13/00
Instructor						Program #	23.5
Date	Start	Stop	Inter. Time	Delta Time	Phase	Comments	
3/13	9:34				Planning	Multi-user calendar extension	

Figure 2.6. Time Recording Log after Cam has started planning for the Multi-user extension project.

Cam follows the PSP2 Planning Script and fills out the PSP2 Project Summary Form. Figure 2.7 shows the first page of the PSP2 Project Summary form.

Having already completed step 1 of the planning script, Program Requirements, he understands the requirements for the project. Cam moves on to step 2 of the PSP2 Planning Script, Size Estimate. He develops a rough conceptual design and fills out the conceptual design form (See figure 2.8).

Once he has his initial design Cam uses the PROBE method to estimate the new and changed lines of code (LOC) required to develop this program. He counts the base size of the calendar project, estimates the number of lines that must be deleted, and estimates the number of lines that will be modified. Then based upon the number of new objects to be added, he calculates the number of new LOC that will be added to the project. Once he has the new LOC he calculates the linear regression parameters  $\beta_0$ ,  $\beta_1$  and  $r^2$  for his estimated LOC and the actual LOC from his previous projects. Table 2.1 shows the values of Cam's previous projects.

**Table C55 PSP2 Project Plan Summary**

Student	Cam Moore	Date	3/13/00
Program	Multi-User Calendar	Program #	23.5
Instructor		Language	Java

Summary	Plan	Actual	To Date
LOC/Hour			
Planned Time			
Actual Time			
CPI(Cost-Performance Index)			
			(Planned/Actual)
% Reused			
% New Reused			
Test Defects/KLOC			
Total Defects/KLOC			
Yield %			
<b>Program Size (LOC):</b>	<b>Plan</b>	<b>Actual</b>	<b>To Date</b>
Base(B)			
	(Measured)	(Measured)	
Deleted (D)			
	(Estimated)	(Measured)	
Modified (M)			
	(Estimated)	(Measured)	
Added (A)			
	(Measured)	(T-B+D-R)	
Reused (R)			
	(Estimated)	(Measured)	
Total New & Changed (N)			
	(Estimated)	(A+M)	
Total LOC (T)			
	(N+B+M+D-R)	(Measured)	
Total New Reused			
Upper Prediction Interval (70%)			
Lower Prediction Interval (70%)			

Figure 2.7. PSP2 Project Summary form after Cam has filled in the header information for the Multi-user extension project.

**Table C39d Conceptual Design Worksheet**

Student	Cam Moore	Date	3/13/00
Instructor		Program #	23.5

Base Class Names:	Calendar	Event	Date	Action
-------------------	----------	-------	------	--------

Reused Class Names:	User			
---------------------	------	--	--	--

Class Name	Method Name	Params	Return Val.
Permission	Permission	User, Level, Event	
Permission	Permission	User, Level	
Permission	Check	Event	boolean

Figure 2.8. Conceptual Design form after Cam has developed a conceptual design for the multi-user project.

Table 2.1. Cam's historical size and time data.

Project	Estimated LOC	Actual LOC	Actual Time
1	702	624	983
2	450	1296	436
3	1340	2252	1369
4	812	2109	1204
5	119	160	111
6	160	231	115
7	160	217	230
8	255	630	119
9	1190	403	539

For his previous project the values are  $\beta_0 = 232.95$ ,  $\beta_1 = 1.12$  and  $r^2 = 0.406$ . Since  $r^2 > 0.50$ , Cam concludes that his prior planned and actual are not highly correlated. Therefore, he will use his original estimate of 1249 for this project. Figure 2.9 shows the Size Estimation form for the multi-user calendar project.

With the Size Estimation Template completed, Cam can fill out the size portion of the Project Summary Form. Figure 2.10 shows the PSP2 Project Summary for the Multi-user Calendar project after Cam fills out the planned size portion.

Cam can now move to step 3 of the PSP2 planning script, Resource Estimation. Cam uses the PROBE method to estimate the time required to develop the new program. To use the PROBE method Cam needs to have his historical data for estimated LOC, actual LOC, and actual time spent. Table 2.1 shows Cam's historical data.

The first step in the method is to determine if the estimated LOC and actual time data are highly correlated. Cam uses his scientific calculator to calculate the linear regression coefficients for the estimated LOC and actual time,  $\beta_0 = 62.154$ ,  $\beta_1 = 0.876$  and  $r^2 = 0.666$ . Since  $r^2$  is greater than 0.5, Cam should use his estimated LOC and actual time data to do the time estimation. Estimated time =  $\beta_0 + \beta_1 * \text{Size Estimate}$ . The estimated time for the Multi-user project is 1244 minutes or about 21 hours. Using the 70% confidence interval, Cam produces an estimation range of 830 minutes to 1658 minutes.

If the estimated LOC and actual time were not highly correlated, Cam would have checked his actual LOC and actual time for correlation. If they were correlated he would use that set of data to calculate the estimated time.

Finally, if neither the estimated LOC nor actual LOC were highly correlated then Cam would use historical averages for productivity using actual LOC and actual time.

Given the estimated time of 1244 minutes and Cam's historical distribution of effort shown in Table 2.2, Cam can fill out the Planned Time portion of the PSP2 Project Summary, see Figure 2.11.

Given his estimated size and estimated time, Cam calculates his estimated productivity, 60.24 LOC/hour and fills out the Planned LOC/hour.

Cam can now move to step 4 of the PSP2 Planning script, Task and Schedule planning. Based upon his historical time data, Cam knows he has three direct hours of work per day for programming tasks. He breaks the Multi-User calendar project into the development phases and fills out the Task Planning Template (see Figure 2.12) and the Schedule Planning Template (see Figure

Table C39a Size Estimating Template

Student	Cam Moore	Date	3/13/00	
Instructor		Program #	23.5	
<b>BASE PROGRAM</b>			LOC	
BASE SIZE (B)	=> => => => => => => => =>		2000	
LOC DELETED (D)	=> => => => => => => => =>		50	
LOC MODIFIED (M)	=> => => => => => => => =>		100	
<b>PROJECTED LOC</b>				
BASE ADDITIONS:	TYPE	METHODS	REL. SIZE	LOC
Event	Data	5	Med	76
TOTAL BASE ADDITIONS (BA)			=> => => => => => => => =>	76
<b>NEW OBJECTS:</b>				LOC (NewReuse*)
Permissions	Data	14	Med	215
Scheduler	Control	20	Med	456
AccessControl	Control	10	Med	152
UserRecognition	Control	23	Med	350
TOTAL NEW OBJECTS (NO)			=> => => => => => => => =>	1173
<b>REUSED OBJECTS</b>				LOC
User				437
REUSED TOTAL (R)			=> => => => => => => => =>	437
Projected LOC:			$P = BA + NO$	1249
Correlation (prior planned vs. actual)			$r^2$	0.4058
Regression Parameter:			$\beta_0$ (use 0 if $r^2 < 0.50$ )	0
Regression Parameter:			$\beta_1$ (use 1 if $r^2 < 0.50$ )	1
Estimated New and Changed LOC:			$N = \beta_0 + \beta_1 * (P + M)$	1349
Estimated Total LOC:			$T = N + B - D - M + R$	3636
Estimated Total New Reused (sum of * LOC):				1173
Prediction Range:			Range	881
Upper Prediction Interval/Largest Likely Size			$UPI = N + Range$	2628
Lower Prediction Interval/Smallest Likely Size			$LPI = N - Range$	866
Prediction Interval Percent:			70% or 90%	70

C39 Instructions: pp. 119-125, 684-685, PSP10.ppt

<sup>1</sup> L-Logic, I-I/O, C-Calculation, T-Text, D-Data, S-Set-up

Figure 2.9. Size Estimation form after Cam has developed a conceptual design for the multi-user project.



**Table C55 PSP2 Project Plan Summary**

Student	Cam Moore	Date	3/13/00
Program	Multi-User Calendar	Program #	23.5
Instructor		Language	Java

Summary	Plan	Actual	To Date
LOC/Hour			
Planned Time			
Actual Time			
CPI(Cost-Performance Index)			
			(Planned/Actual)
% Reused			
% New Reused			
Test Defects/KLOC			
Total Defects/KLOC			
Yield %			
<b>Program Size (LOC):</b>	<b>Plan</b>	<b>Actual</b>	<b>To Date</b>
Base(B)	2000		
	(Measured)	(Measured)	
Deleted (D)	50		
	(Estimated)	(Measured)	
Modified (M)	100		
	(Estimated)	(Measured)	
Added (A)	1249		
	(Measured)	(T-B-D-R)	
Reused (R)	437		
	(Estimated)	(Measured)	
Total New & Changed (N)	1349		
	(Estimated)	(A+M)	
Total LOC (T)	3636		
	(N+B+M-D-R)	(Measured)	
Total New Reused			
Upper Prediction Interval (70%)	2628		
Lower Prediction Interval (70%)	866		

Figure 2.10. PSP2 Project Summary form after Cam has filled in the planned size information for the Multi-user extension project.

Table 2.2. Cam's historical time distribution.

Phase	%
Planning	2
Design	20
Design Review	2
Code	34
Code Review	3
Compile	7
Test	29
Postmortem	3

Time in Phase (min.)	Plan	Actual	To Date	To Date %
Planning	24			
Design	249			
<i>Design review</i>	25			
Code	419			
<i>Code review</i>	50			
Compile	84			
Test	359			
Postmortem	34			
Total	1244			
<i>Total Time UPI (70%)</i>	1658			
<i>Total Time LPI (70%)</i>	830			

Figure 2.11. PSP2 Project Summary form after Cam has filled in the planned time information for the Multi-user extension project.

2.13). If all goes according to plan Cam will be finished with the Multi-User calendar project in a week and a half.

**Table C47a Task Planning Template**

Student	Cam Moore	Date	3/13/00
Project	Multi-User Calendar	Instructor	

Task		Plan					Actual		
#	Name	Hours	Planned Value	Cumulative Hours	Cumulative Planned Value	Date Monday	Date	Eamed Value	Cumulative Eamed Value
1	Planning	0.42	2.2	0.42	2.2	3/13			
2	Design	4.15	21.4	4.57	23.6				
3	Design Rev.	0.42	2.2	4.99	25.8				
4	Code	6.98	36.0	11.97	61.8				
5	Code Rev.	0.83	4.3	12.80	66.1				
6	Test	5.98	31.0	18.78	97.1	3/20			
7	Postmortem	0.56	2.9	19.34	100				

Figure 2.12. PSP2 Task Planning Template after Cam has filled in the planned task information for the Multi-user extension project.

After finishing the task and schedule phase, Cam can now move to the last step in the PSP2 Planning Script, Defect Estimation. Based upon his historical data on defects per LOC, Cam injects 98 defects per thousand lines of code (KLOC). He estimates the total defects that he will find in the Multi-User calendar program will be 120. Based upon his historical defect distribution (see Table 2.3), Cam calculates the estimated number of defects he will inject and remove per phase (see

Table 2.3. Cam's historical defect distribution.

Phase	Injected%	Removed%
Planning	0	0
Design	12	0
Design Review	0	2
Code	88	9
Code Review	0	6
Compile	0	57
Test	0	23
Postmortem	0	4

Table 2.4) and inputs these numbers on the PSP2 Project Plan Summary (see Figure 2.14).

**Table C49a Schedule Planning Template**

Student	Cam Moore	Date	3/13/00
Project	Multi-User Calendar	Instructor	

		Plan			Actual			
Week No.	Date	Direct Hours	Cumulative Hours	Cumulative Planned Value	Direct Hours	Cumulative Hours	Cumulative Earned Value	Adjusted Earned Value
1	3/13	15	15	77.5				
2	3/20	4.34	19.34	100.0				

Figure 2.13. PSP2 Schedule Planning Template after Cam has filled in the planned schedule information for the Multi-user extension project.

Table 2.4. Cam's planned defect distribution.

Phase	# Injected	# Removed
Planning	0	0
Design	14	0
Design Review	0	2
Code	106	11
Code Review	0	7
Compile	0	68
Test	0	27
Postmortem	0	5

**Table C55 PSP2 Project Plan Summary (continued)**

Student	Cam Moore	Date	3/13/00
Program	Multi-user Calendar	Program #	23.5
Instructor		Language	Java

<b>Defects Injected</b>	<i>Plan</i>	<b>Actual</b>	<b>To Date</b>	<b>To Date %</b>
Planning	0			
Design	14			
<i>Design review</i>	0			
Code	106			
<i>Code review</i>	0			
Compile	0			
Test	0			
Total Development	120			
<b>Defects Removed</b>	<i>Plan</i>	<b>Actual</b>	<b>To Date</b>	<b>To Date %</b>
Planning	0			
Design	0			
<i>Design review</i>	2			
Code	11			
<i>Code review</i>	7			
Compile	68			
Test	27			
Total Development	115			
After Development	5			
<i>Defect Removal Efficiency</i>	<i>Plan</i>	<i>Actual</i>	<i>To Date</i>	
<i>Defects/Hour - Design review</i>	4.8			
<i>Defects/Hour - Code review</i>	8.4			
<i>Defects/Hour - Compile</i>	48.6			
<i>Defects/Hour - Test</i>	4.5			
<i>DRL(DLDR/UT)</i>	1.07			
<i>DRL(CodeReview/UT)</i>	1.87			
<i>DRL(Compile/UT)</i>	10.8			

Figure 2.14. PSP2 Project Plan Summary after Cam has finished planning the defect portion of the Multi-user Calendar extension project.

Cam is finished with the planning phase so he notes the time in the Time Recording log and calculates the elapsed time (see Figure 2.15).

**Table C16a Time Recording Log**

Student		Cam Moore				Date	3/13/00
Instructor						Program #	23.5
Date	Start	Stop	Inter. Time	Delta Time	Phase	Comments	
3/13	9:34	10:02		28	Planning	Multi-user calendar extension	

Figure 2.15. Time Recording Log after Cam has finished planning the Multi-user extension project.

Cam schedules a meeting with Philip for the next day. At the meeting he tells Philip that the Multi-User calendar extension should take between a week (lower prediction interval) and two weeks (upper prediction interval) to complete. Cam's best estimate is that it will take a week and a half. Philip agrees that this is a reasonable schedule for the project.

**Design** Cam returns to his cubicle and starts using the PSP2 Development script to build the Multi-User calendar program. The first step is Design. He enters the start time on the TRL. Using the conceptual design he came up with during the Planning phase, Cam fleshes out the design. While he is working on the design his phone rings. He notes the time on a slip of paper and answers the call. When he finishes the phone call he notes the time and records the number of minutes spent as interrupt time on the TRL. Cam stops working on the design to go to a staff meeting so he records the stop time in the TRL (see Figure 2.16).

**Design Review** After Cam finishes the design of the Multi-User calendar program he takes out the task planning form and fills out the actual columns. Then he starts the next step of the PSP2 development script, Design Review. He takes his Design Review Checklist (see Figure 2.17) and the Multi-User calendar program design and conducts an individual review. When Cam finds a defect in the design he records the defect on the PSP2 Defect Recording Log (DRL). Once he finishes the review he goes back and fixes all the defects he finds. While he is fixing the defects he records the amount of time it takes to fix each defect. For example, the loop error, defect number 1, took only 1 minute to fix (see Figure 2.18).

**Table C16a Time Recording Log**

Student		Cam Moore				Date	3/13/00
Instructor						Program #	23.5
Date	Start	Stop	Inter. Time	Delta Time	Phase	Comments	
3/13	9:34	10:02		28	Planning	Multi-user calendar extension	
3/14	9:12	10:55	26	67	Design		

Figure 2.16. Time Recording Log after Cam goes to the staff meeting.

**Table C57 Java PSP2 Design Review Checklist**

PROGRAM NAME AND #: Multi-User calendar 23.5

Purpose	To guide you in conducting an effective design review				
General	As you complete each review step, check that item in the box to the right. Complete the checklist for one program unit before you start to review the next.				
Complete	Ensure that the requirements, specifications, and high-level design are completely covered by the design: - all specified outputs are produced - all needed inputs are furnished - all required includes are stated				
Logic	Verify that program sequencing is proper: - that stacks, lists, etc. are in the proper order - that recursion unwinds properly Verify that all loops are properly initiated, incremented, and terminated				
Special Cases	Check all special cases:				

Figure 2.17. Cam's Design Review Checklist for Java.

**Table C18a Defect Recording Log**

Name: Cam Moore					Program: Multi-user calendar		Page: 1
Date	No.	Type	Inj.	Rem.	Fix Time	Fix Def.	Description
3/15	1	logic	Des	Des r	1		Loop incorrect

Figure 2.18. Cam's Defect Recording Log showing one design defect found in the design review. It took him one minute to fix the defect.

**Code** Once all the defects found in the design review are fixed, Cam enters the stop time in the TRL, updates the Task Planning Template with the actual times, and moves on to the third step in the PSP2 development script, code. While Cam is coding the program, he records each requirements

**Table C16a Time Recording Log**

Student		Cam Moore				Date	3/13/00
Instructor						Program #	23.5
Date	Start	Stop	Inter. Time	Delta Time	Phase	Comments	
3/13	9:34	10:02		28	Planning	Multi-user calendar extension	
3/14	9:12	10:55	26	67	Design		
3/14	1:10	3:25	15	120	Design		
3/15	9:02	9:55		53	Design		
3/15	10:20	10:56		36	Design Rev.		
3/15	10:56				Code		

Figure 2.19. Time Recording Log after Cam has finished his design review and started coding.

and design defect he finds and the amount of time it takes him to fix it. For example, when he realizes that the Permission class needed an additional public method to set the level he stops coding and pulls out the DRL for the project, finds a pencil and records the defect. Then he fixes the problem remembers how long it takes to fix. Then he goes back to the DRL to fill in the fix time as shown in Figure 2.20.

As Cam is coding the program he must be aware of interruptions and record them in the TRL. If his phone rings or a colleague comes into ask him a question he must time the interruption and record it in the TRL. At the end of the week Cam takes out the Schedule Planning Template and fills out the Actual hours and earned values for the project.



**Table C18a Defect Recording Log**

<b>Name:</b> Cam Moore					<b>Program:</b> Multi-user calendar		<b>Page:</b> 1
Date	No.	Type	Inj.	Rem.	Fix Time	Fix Def.	Description
3/15	1	logic	Des	Des r	1		Loop incorrect
3/15	2	Miss	Des	Des r	2		Missing return value from method.
3/16	3	Miss	Des	Code	3		Missing method for setting level in Permission

Figure 2.20. Cam's Defect Recording Log showing the design defect found in the design review and the design defect found during coding. It took him three minutes to fix the second defect.

**Code Review** When Cam finishes coding the entire project, he enters the stop time in the TRL, updates the Task Planning Template, and moves on to the next phase in the PSP2 Development Script, Code Review. He finds his Code Review Checklist, gets out his DRL and then enters the start time for the code review in the TRL. He then reviews all the code in the project. When he finds a defect in the code he records the defect in the DRL then fixes the defect. Again when he is fixing the defect he must record how much time it takes to fix the defect in the DRL. When he finishes reviewing all the code he records the stop time in the TRL and updates the Task Planning template by entering the actual earned value.

**Compile** Cam can now move on to the fifth step in the PSP2 Development Script, Compile. He records the start time in the TRL and runs the compiler for the first time. For each defect the compiler finds he records it in the DRL and fixes it. Again he records how long it takes him to fix all the errors. When he finally gets a clean compile he is finished with the compile phase. He enters the stop time in the TRL and updates the Task Planning Template.

**Test** Cam moves to the sixth and final step in the PSP2 Development Script, Test. He records the start time in the TRL and starts running all his tests. For each test he runs he fills out a Test Report Template (see Figure 2.21). For each defect found during test Cam stops work and records the defect in the DRL, recording how long it takes him to fix the defect. For each defect he must determine which phase the defect was injected into the program. When the program passes all the tests, Cam records the end time of testing, updates the Task Planning Template and moves to the last phase of the PSP2 Process Script, Postmortem.

**Table C37a Test Report Template**

Student	Cam Moore	Date	3/20/00
Instructor		Program #	23.5

Test Name	Constructor	Number:	1
Test Objective	Test correct construction of all objects		
Test Description	Creates an instance of each new object and determines if they are correct		
Test Conditions			
Expected Results	One new instance of each new class		
Actual Results	One new instance of each new class		
Test Name		Number:	

Figure 2.21. Cam's Test Report Template showing the first test that tests all the constructors. The test eventually passed.

**Postmortem** During the Postmortem phase Cam will complete the PSP2 Project Summary Form for the Multi-User project. Cam records the start time for the Postmortem on his TRL. He then gathers the Project Plan Summary, the Task Planning Template, the Schedule Planning Template, the completed Test Report Templates, the completed Design and Code Review Checklists, the completed Time Recording Log, the Completed Defect Recording Log and the tested, running program.

The first step of the Postmortem Script is to count the number of defect injected during each phase of development. Cam takes out the Defect Recording Log and counts all defects injected for each phase. He enters these numbers under Defects Injected — Actual on the Project Plan Summary. Cam looks at the last project's Project Plan Summary to find out the To Date Defects Injected and adds this project's numbers to those numbers to fill out the To Date and To Date % columns. Cam notices that he did not record as many defects as he planned. As he reflects on the project, he remembers not recording all the compile defects that he found. He did not want to interrupt his programming to record all the syntax errors on the Defect Recording Log. He just recorded the ones that he thought were important.

The second step is to count the number of defects removed during each phase of development. Cam uses the Defect Recording Log and counts all the defects removed for each phase. He enters these numbers under Defects Removed — Actual on the Project Plan Summary. By looking at the last project's Project Plan Summary, Cam fills out the To Date and To Date % columns for defects removed (see Figure 2.22). He then calculates the Yield for this project. The Yield for the project is the number of defects removed before the compile phase divided by the total number of defects removed.

**Table C55 PSP2 Project Plan Summary (continued)**

Student	Cam Moore		Date	3/13/00
Program	Multi-user Calendar		Program #	23.5
Instructor			Language	Java

<b>Defects Injected</b>	<i>Plan</i>	<b>Actual</b>	<b>To Date</b>	<b>To Date %</b>
Planning	0	0	0	0
Design	14	10	23	15
<i>Design review</i>	0	0	0	0
Code	106	32	130	85
<i>Code review</i>	0	0	0	0
Compile	0	0	0	0
Test	0	0	0	0
Total Development	120	42	153	
<b>Defects Removed</b>	<i>Plan</i>	<b>Actual</b>	<b>To Date</b>	<b>To Date %</b>
Planning	0	0	0	0
Design	0	0	0	0
<i>Design review</i>	2	2	4	1
Code	11	7	17	11
<i>Code review</i>	7	7	14	9
Compile	68	20	83	56
Test	27	6	31	22
Total Development	115	42	149	
After Development			4	
<i>Defect Removal Efficiency</i>	<i>Plan</i>	<i>Actual</i>	<i>To Date</i>	
<i>Defects/Hour - Design review</i>	4.8	3.3		
<i>Defects/Hour - Code review</i>	8.4	6.7		
<i>Defects/Hour - Compile</i>	48.6	8.2		
<i>Defects/Hour - Test</i>	4.5	0.9		
<i>DRL(DLDR/UT)</i>	1.07	3.7		
<i>DRL(CodeReview/UT)</i>	1.87	7.4		
<i>DRL(Compile/UT)</i>	10.8	9.1		

Figure 2.22. Cam's Filled in PSP2 Project Plan Summary form after counting all the recorded defects.

The third step of the the PSP2 Postmortem Script is to count the actual size of the program. Cam uses LOCC[28] to determine the base size of the project, the number of reused LOC and the new LOC. Cam is unable to count the modified lines of code because he does not have any automated support to determine which lines have been modified. He enters the data on the Project Plan Summary (see Figure 2.23).

<b>Program Size (LOC):</b>	<b>Plan</b>	<b>Actual</b>	<b>To Date</b>
Base(B)	2000	2000	
	(Measured)	(Measured)	
Deleted (D)	50		
	(Estimated)	(Counted)	
Modified (M)	100		
	(Estimated)	(Counted)	
Added (A)	1249		
	(Measured)	(T-B+D+R)	
Reused (R)	437	314	
	(Estimated)	(Counted)	
Total New & Changed (N)	1349	2139	
	(Estimated)	(A+M)	
Total LOC (T)	3636	4453	
	(T-B+M+D+R)	(Measured)	
Total New Reused			
<i>Upper Prediction Interval (70%)</i>	2628		
<i>Lower Prediction Interval (70%)</i>	866		

Figure 2.23. Cam's Filled in PSP2 Project Plan Summary form after counting the program size with LOCC.

The fourth step of the PSP2 Postmortem Script is to add up all the time spent for each phase of development. Cam takes out his TRL and adds up all the time for each phase. He then enters the numbers on the Project Plan Summary (see Figure 2.24). Since he now has both actual size and time he can calculate the actual LOC/Hour and the Cost-Performance Index (CPI). The CPI is the Planned time/Actual time or 0.92 for this project. This ratio is intended to give Cam an indication of how well he is planning his projects. For this project, Cam slightly under-planned, 8%, how long the project will take. Cam is very happy with this level of accuracy.

Now that he has the time spent per phase and the defects removed per phase, Cam calculates the actual Defect Removal Efficiency. He then calculates the Defect Removal Leverage. The defect removal leverage provides a useful measure of the relative effectiveness of various defect removal methods[17]. It is the ratio of the defects removed per hour in any two phases. In PSP2, the

<b>Time in Phase (min.)</b>	<b>Plan</b>	<b>Actual</b>	<b>To Date</b>	<b>To Date %</b>
Planning	24	28	145	2
Design	249	240	1220	18
<i>Design review</i>	25	36	136	2
Code	419	432	2434	35
<i>Code review</i>	50	62	335	5
Compile	84	145	565	8
Test	359	380	1810	26
Postmortem	34	24	192	4
Total	1244	1347	6837	
<i>Total Time UPI (70%)</i>	1658			
<i>Total Time LPI (70%)</i>	830			

Figure 2.24. Cam's Filled in PSP2 Project Plan Summary form after adding up all the time entries from the Time Recording Log.

efficiency of Design Reviews, Code Reviews, and Compile are compared to the efficiency of Test. The values, 3.3, 6.7, and 8.2 show Cam that removing defects early in the development cycle are much more efficient than removing defects during test. The high value of 8.2 for Compile makes Cam believe that most of the defects that he removes in Compile are just simple syntax errors and not very interesting.

#### 2.1.4 Evaluations of the PSP

Since it has been introduced in 1995, many researchers have conducted studies on the effectiveness and benefits of the PSP. The results of these studies are mixed. Some studies show great improvement in the quality of the software developed and improvements in the productivity of the software engineers. Other studies show no improvement in the software developers and low adoption of the PSP in industry.

In a 1996 article, Watts Humphrey reported the results of 104 engineers taking the PSP course[18]. He collected the PSP data for his students learning the PSP. The students recorded their time, size and defect data for the 10 programming assignments in the PSP course. His subjects were a mix of software professionals and college students. Based upon the data from the students, he states that the two goals of PSP were met.

First, reported defects fell from an average of 116.4 defects per thousand LOC (KLOC) for assignment 1 to 48.9 defects per KLOC for assignment 10. The total number of defect was reduced by 59.8% and the test defects were reduced by 73.2%. These reductions shows an impressive improvement in the quality of the software as measured by self-reported defects and defect density. The defects reported are the defects found during development not defects found after development. The engineers also improved their lines of code written per hour by an average of 20.8%. The students, on average, reduced the time they spent compiling by 81.7% and reduced testing time by 43.3%.

Second, the estimation accuracy of the students increased. For assignment 1, 32.7% of the engineers' estimates were within 20% of their actual times. By assignment 10, 49.0% of the engineer's estimates were within 20% of their actual times. The engineers reduced size-estimation errors by 25.8% and time-estimation errors by 40%. These results show that the students in the course did get better at estimating the amount of time it would take them to complete their assignments. For the first three assignments the students make a "best guess" of the project time. They do not have enough historical data to allow them to use the time estimation technique that the PSP uses.

In 1996, Sherdil and Madhavji studied human-oriented improvement in the Software Process[34]. They also used data from teaching the PSP as a basis for their studies. They found that the students' defect density dropped by 13% after project 6, when code reviews are introduced. One implication is that the introduction of design and code reviews helped reduce the number of defects by 13%. They also found that their subjects reduced their size estimation error by more than 7%. Since size estimation is key to time estimation, this implies that the students were getting better at planning their projects.

Hayes and Over conducted an extensive study, with 298 engineers, of the PSP[14]. The results of the study were impressive. Over the projects completed, the median improvement in size estimation was a factor of 2.5. This means that 50% of the engineers reduced their size estimation error by a factor of 2.5. The median improvement in time estimation was 1.75. The median reduction in overall defect density was by a factor of 1.5. The engineers substantially reduced the percentage of defects surviving to later stages of development.

Pat Ferguson and others report excellent results with PSP adoption at Advanced Information Services, Motorola and Union Switch and Signal[9]. Their study was of three industrial software engineering groups from three different companies. Each group was trained in the PSP

and used it to develop projects for normal operations. The PSP data from these project was collected for the study.

At Advanced Information Services, they compared several different projects. In a comparison of schedule performance, projects from the same development team before PSP training and after PSP training were compared. Before the PSP, the average schedule estimating error was 394 percent. After using the PSP, the average schedule estimating error was -10.4 percent. They also compared the defect density between two group of engineers, one group that had the PSP training and another group, working on the same project, that didn't have the PSP training. The defect densities were calculated by the number of defects found in acceptance testing. Acceptance testing was done by an external group. Before the PSP training the two groups had roughly the same defect density as measured by defects found in acceptance testing of .8 defects per 1000 lines of code. After PSP training the trained group's defect density fell to 0.17 defects per 1000 lines of code, an improvement of 78 percent. The productivity of the PSP trained engineers was compared to their productivity before training and the other, non trained, group. Their post PSP productivity increased by 7.4 percent.

At Motorola paging product group, three PSP training classes were conducted. The training and use of PSP was supported by Motorola. After the course a survey indicated that most of the engineers would continue to use the PSP in their development. Another survey six months later showed that more than 80 percent of the engineers used PSP methods in their work. Eighteen projects were completed by PSP-trained engineers. Most of the projects were maintenance projects of working paging systems. At the time of the report only one use defect was found in any of the eighteen projects.

At Union Switch & Signal, PSP trained engineers worked on five project. All were maintenance and enhancement release for a large railroad information and control system. As of the date of the report no use defects had been reported in any of the projects.

In all three of these case studies the benefits of the PSP were shown through external measures of schedule performance and usage defects. These studies show good adoption of aspects of the PSP, improved schedule estimation, and strong quality improvements in the developed software.

Another study of industrial usage of the PSP conducted by Barry Shostak and others reports poor adoption of PSP in industry[35, 7]. They conducted a study of engineers at CAE Electronics Ltd. They studied 28 software engineers who were taught the PSP. The researchers modified the original PSP training to account for the software engineers' working environment. They reduced the training time since spending 70 hours of training time and 150 to 200 hours for

assignments is not possible for working engineers. They surveyed some engineers that had taken the PSP training from McGill University to find out why they were not using the PSP in their work. The reasons that the engineers didn't use the concepts from the PSP were:

- Programming tasks in industry are usually conducted in teams. It is not obvious how the individual tasks in the PSP could be applied to team work.
- The collection of all the PSP data with out automated support was too cumbersome. The engineers did not have the appropriate tools to support the PSP.
- There was a lack of management support. The engineers felt that the overhead of data collection and analysis would have to come from their own time and since they were putting in overtime they did not want to spend more time.
- There is a hero culture in the organization. Programmers who are more disciplined and produce code with less defects are less respected than the troubleshooting heroes who save projects in crises due to inadequate programming practices.

The researchers also addressed some of the issues with the data collection forms. The following are some of the issues that they had to address:

- There was some mismatch between the personal process model of the PSP and the process used by the engineers. The PSP uses a development model, while some of the engineers did mainly maintenance work. This forced the researchers to modify the forms.
- After modifying the forms they needed to test them out. There were some practical considerations like making copies of two sided forms with a single sided copier and a misunderstanding of the terminology used on the forms.
- Some engineers did not want to fill out the defect collection form for all the syntax errors that they made. The researchers modified the defect recording log to have a checklist for some commonly made defects.

Their study found several issues with the engineers' use of the PSP. Some of issues that the authors said need addressing are:

- There is a need for constant feedback to reinforce the learning.



- Many participants felt that automated data collection and analysis also would have been useful. When the researchers looked into implementing automated tools they found that the number of different platforms used by the engineers would be a substantial constraint. They also found the need for a common lines of code counter to ensure that the size measurements are consistent.
- Continued support for using the PSP was very important.
- Some of the engineers felt that they spent very little of their time actual programming and that interruptions took up too much of their time.

During the study the researchers found that only 46.5% of the engineers kept using concepts from the PSP seven months after the end of the training.

Many of the studies on the PSP assumed that the data recorded and analyses made by the subjects using the PSP was accurate and correct. Anne Disney conducted a study to see if this assumption was correct.

### **2.1.5 Disney Thesis on Data Quality in the PSP**

After teaching and learning the PSP for two years, Dr. Philip Johnson and Anne Disney started to notice some common trends in the student's PSP data. Dr. Johnson had his students conduct reviews of each others PSP forms before they turned them in. During these review some defects were found. Some of these trends and their own use of the PSP in their own development made them think about the issue of data quality in the PSP.

Anne Disney did her masters thesis on data quality issues in the PSP. She took the PSP forms from Dr. Johnson's PSP class and entered in all the values in to her automated PSP tool (See section 2.2.1.3 for a description). Her PSP Tool automated all the PSP analysis and compared the students' numbers to the correct calculated numbers. She found that in her sample of students who learned the PSP, the errors in their data were significant. The errors in data collection and analysis lead to incorrect insights into the students development practices. For example, in several cases the students' incorrect data indicated that they were over estimating their yield when in fact they were underestimating their yield. Thinking that they were over estimating their yield when they were actually underestimating, could lead to process changes that could harm their software process.

In her thesis, Disney classified the data errors found in the PSP data into seven categories:

- **Calculation Error:** This error type applied to data fields whose values were derived using any sort of calculation and the calculation is done incorrectly. In her study 46% of all the errors were calculation errors.
- **Blank Fields:** This error type applies to data fields that are required but not filled in. 18% of all the errors in the study were blank fields.
- **Inter-Project Transfer Error:** This error type applies to data fields whose values involved data from a prior project and the value is not the same as the prior project's value. Inter-project transfer errors accounted for 14% of the errors in Disney's study.
- **Entry Error:** This error type applies to fields where the user clearly does not understand the purpose of the field or used an incorrect method in selecting data. 9% of the errors were entry errors.
- **Intra-Project Transfer Error:** This error type applies to data fields whose values involve other data fields in the same project, but are incorrectly filled in. 6% of the errors were intra-project transfer errors.
- **Impossible Values:** This error type indicates that two values were mutually exclusive. 6% of the errors were impossible values.
- **Sequence Error:** This error type is used to indicate when the user moved back and forth between phases. Only 1% of the errors were sequence errors.

She found that 34% of the errors made affected multiple forms and multiple projects. This means that an error in an earlier project rippled through the future projects affecting the student's PSP data. One possible solution is automated tool support to reduce data errors. Human beings will make mistakes in any process. Automating much of the data entry and transfer will reduce the opportunity to make mistakes.

Another important finding of her research was that automated support for the PSP is required for successful adoption and use of the PSP. The problems of data quality in the PSP require automated data collection and analysis. Without automated support accurately recording, transferring and analyzing all the data is too difficult.

### **2.1.6 The PSP and LEAP**

Project LEAP and the Leap toolkit are based upon the PSP. The Leap toolkit collects the same metrics about the development process as the PSP. Both focus on individual developer improvement. Many of the analyses that the Leap toolkit performs come directly from the PSP.

Project LEAP and the Leap toolkit differ from the PSP in many ways. First, I noticed that the PSP requires the user to collect size, time and defect data about all their projects. I feel that valuable insights can still be gained without collecting all this data. Developers interested in their planning and estimating skills should not have to collect defect data. Second, I feel that the PSP's waterfall development model is too inflexible. There are many different development models used by developers. The developer should use the development process that is best for them and the development situation. Third, the PSP is tailored for code development. Software developers do more than produce code. A tool for software developer improvement should support the other work that developers do. Empirical individual process improvement can apply to many different processes other than code development.

Besides being flexible, LEAP tools must reduce the overhead to the user. Since the PSP is a manual process, it creates additional overhead. To reduce the overhead of collecting and analyzing the data, the Leap toolkit provides several tools for data collection and analysis. Not only does the Leap toolkit reduce the overhead to the user, but by automating the collection and analysis of the data, it eliminates many of the data errors that Disney found.

There are several other tools that automate the whole PSP or data collection. The next section discusses these tools.

## **2.2 Automated PSP Tools**

Soon after the PSP was introduced many developers built tools to support data collection and analysis for the PSP. Some developers automated the entire PSP while others just automated different aspects of the PSP.

### **2.2.1 Full automation**

#### **2.2.1.1 psptool**

psptool by Andrew M. Worsley[32] is a tool that runs under X/Unix or on Win32S platforms. It allows the user to collect size, time and defect data. It also produces a PSP2.1 like plan

summary and supports time estimation based upon historical data and an initial size estimate. This tool does not directly support any of the other PSP levels. The analyses that it performs are focused on PSP2.1.

#### **2.2.1.2 PSP Studio**

PSP Studio from Eastern Tennessee State University's Design Studio 1997[15] automates all the PSP levels. It runs on Win32 platforms and supports all the PSP levels from 0 through 3.0. It produces all the postmortem reports after the projects are complete.

The major benefit of this tool is that it faithfully supports the PSP processes. If a person is trained in the PSP using PSP Studio is very simple and straight forward. Since it handles all the data manipulation many of the data transfer and calculation issues are solved.

Some of the drawbacks of the tool are data storage, and lack of flexibility. The tool stores its data in a database. Storing the data in a database simplifies data access, but makes data transfer more difficult. PSP Studio does not allow the user to modify the processes used. The user must choose one of the seven PSP level to develop their software.

#### **2.2.1.3 PSP Tool**

PSP Tool[4] from Anne Disney, is written in Progress 4GL/RDBMS and runs on SCO Unix. It implements the PSP0, PSP0.1, and PSP1 completely while the higher levels are not fully implemented. The PSP Tool allows the user to define their own *Defect models*. *Defect models* refer to specific defects within a defect type. The user may enter a defect model in the defect recording tool and it will fill in the fields. This reduces the mental overhead of the user and speeds up defect recording. As with the PSP Studio the PSP Tool allows the user to follow one of the implemented PSP levels.

### **2.2.2 Partial PSP automation**

Developers using the PSP record three types of primary data: time, size and defects. Many tool developers have developed tools to automate the collection of one or more of these primary metrics. The following tools focus on collecting the raw data and not enforcing the entire PSP process.

Individuals interested in recording their data may choose any combination of these tools. One major issue of using these different tools is the format of the data recorded by the tools. To

produce a report of defect density or productivity the developer may have to analyze two or more very different data files.

#### **2.2.2.1 pplog-mode, PPLog Control, Timmie and makelog**

Researchers at the University of Karlsruhe developed several tools that help automate the collection of time and defect information[31]. Their data collection tool, pplog-mode.el is an extension for GNU Emacs[12]/XEmacs[42], powerful text editors used by programmers. The developer using these tools can record their time and defects that they find while using Emacs. Users defines a *logging key*. When the *logging key* is pressed Emacs automatically switches to the *logging buffer* where the user may type in a description of the event that just occurred. Emacs automatically inserts the time-stamp of the event. The data is saved in a database file. To analyze the data files the researches wrote a PERL script called evalpsp.

The researchers wrote additional tools for recording data in formats that evalpsp could analyze. They are PPLog Control, a full-featured GUI application for win32 machines, Timmie, a multi-day, multi-project Java application for recording time and defect data, makelog, a command line program for PC users similar to pplog-mode.el. These tools produce data files with a fixed format that can be processed or created by different tools. This method of data storage allows the developers to produce more tools that support the data format.

One major drawback of these tools with respect to wide spread adoption is their reliance on the Emacs editor. While being a very powerful editor, Emacs is not very widely used and has a very steep learning curve. If a developer does not use Emacs for their development then these tools are not useful.

#### **2.2.2.2 titrax**

titrax[36] is a time tracker by Harald T. Alvestrand. It is written in C and runs under X. It allows the user to record their times and includes some simple time analysis tools. These tools since they only run under X are for the Unix developer.

#### **2.2.2.3 PC LOC Accounting Tools**

PC LOC Accounting Tools[31] by Christian Segor are three tools one that inserts tags into source code, one to count the LOC LOC, and one to remove the tags from the source code. The counter is able to count base LOC, modified LOC, added LOC and deleted LOC.

#### **2.2.2.4 locdelta**

locdelta[31] is a perl script that calls a user supplied program to format the source code then calls the Unix diff program to count the base, modifies, added and deleted LOC. Since, this tool requires the Unix diff program it is not available for Windows platforms.

#### **2.2.2.5 LOCC**

LOCC[28] written by Joe Dane is an extensible system for producing hierarchical, incremental measurements of work product size. LOCC can produce output files that the Leap toolkit can use.

### **2.2.3 Automated PSP tools and LEAP**

The Leap toolkit builds upon the ideas of the PSP. It can record all the data needed in the PSP and yet, it does not require the user to record all three types of data for interesting analyses. The Leap toolkit is more flexible than the fully automated PSP support tools, but it does not enforce the PSP processes like they do. The Leap toolkit currently does not support all the reports that the PSP processes require. I can easily add these reports to the Leap toolkit if users want them. The Leap toolkit allows the developer to easily collect and analyze size, time and defect data. By handling all three types of data, the Leap toolkit is more powerful than the partial PSP tools. Since all the data is collected by the Leap toolkit, the user can easily analyze different data types. It is difficult to combine time data from `titrax` and `PC LOC Accounting Tools` to automatically produce productivity data. The Leap toolkit integrates this analysis into the project viewer `Hee`.

The next section discusses the second source of inspiration for LEAP, Formal Technical Review.

## **2.3 Formal Technical Review**

Formal Technical Review is defined as

a method involving a structured encounter in which a group of technical personnel analyzes an artifact according to a well-defined process. The outcome is a structured artifact that assess or improves the quality of the artifact as well as the quality of the method.[10]

The technical personnel that analyze the work product may fulfill many different roles. The generic roles in any FTR are author, moderator, reviewer, scribe, and leader. The author is the

person who created the artifact under review. The moderator moderates the group meetings that may be held during the review process. The reviewers are the technical people who analyze the work product. The scribe records all the issues found by the reviewers. The leader organizes the entire process. During any review an individual may perform many of these roles.

### 2.3.1 The FTR Process

All formal technical reviews follow the same generic process. Figure 2.25 shows the generic FTR process. Many organizations modify the generic process, but it is the basis for FTR.

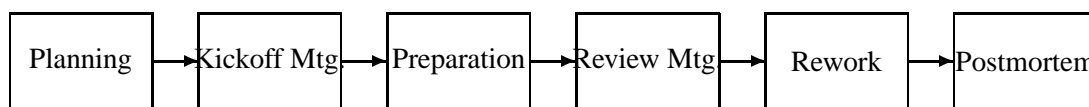


Figure 2.25. Generic Review Process

In the planning phase the review leader plans the review. They gather the review materials: work product, guidelines, checklists, standards, etc. They choose the review members and schedule the meetings and deadlines. Another important part of the planning phase is to determine the goals of the review. The goals of the review help determine the level of formality and the process to use.

Once the planning is done a Kickoff meeting is often held to orient all the review members to the goals of the review and distribute the review materials. The Kickoff meeting is often not used if the review members are familiar with the work product and review process.

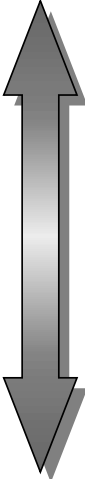
In the preparation phase the reviewers familiarize themselves with the work product. In some review methods like Inspection[8], the reviewers do not record any issues, but just become familiar with the work product. In other methods like FTArm[24, 25, 20, 21, 22, 37] the reviewers record their issues.

In the Review Meeting phase the reviewers gather to discuss the work product. Often the moderator proceeds through the work product and the reviewers raise any issues they have with the work product. The output of the Review Meeting phase is a consolidated list of all the issues found by the reviewers.

During the Rework phase the author of the work product takes the consolidated list of issues and addresses each issue. Some issues may require rework, others may not be defects. The author fixes all the defect that they can.

In the last phase, Postmortem, the review team evaluates the entire review process including the reworked work product. Often the review team approves the work product or decides that it should be re-reviewed. The review process is also analyzed to generate suggestions for improvement.

### 2.3.2 The Spectrum of Reviews



Method Family	Typical Goals	Typical Attributes
<b>Walkthroughs</b>	Developer training Quick turnaround	Little/no preparation Informal process No measurement
<b>Technical Reviews</b>	Requirements elicitation Ambiguity resolution Team building	Formal process Author presentation Wide range of discussion
<b>Inspections</b>	Detect and remove defects efficiently and effectively	Formal process Checklists Measurements Verify phase

Figure 2.26. Spectrum of Formal Technical Reviews

The most informal reviews are called Walkthroughs[44]. In walkthroughs there is very little preparation. The members in the walkthrough gather together and the author walk the group through the artifact explaining what the artifact does. As the author walks through the artifact the reviewers are looking for problems in the artifact. When a problem is discovered, it is often fixed right there in the walkthrough. Walkthroughs often combine the Kickoff meeting, Review meeting, Rework phase, and Postmortem phase all into one meeting. The author is often the moderator, leader and scribe.



More formal reviews are known as Technical Reviews. Technical Reviews are more formal and normally follow all six phases of the review process. The goals of technical reviews may not focus purely on evaluating the work product and can include team building and developer education.

The most formal reviews are Inspections[8]. Inspections have one primary goal, to detect and remove defects from the work product effectively and efficiently. To accomplish this goal the process is very formal and discussion during the Review Meeting is solely focused on reporting defects not solutions to defects. The moderator must control the meeting to ensure that the discussion does not wander.

In the PSP Watts Humphrey introduced the concept of a single person review. The developer conducts a technical review of their work product to detect defects and improve the quality of the work product. The PSP reviews are formal and similar to Inspections since the developer uses checklists to help guide their focus. Combining the PSP's personal reviews and group reviews should help improve the work product and help improve the developer. The defects that the group detects can be analyzed to provide additional insights for the developer.

### **2.3.3 Formal Technical Review and LEAP**

The Leap toolkit supports group review of work products by allowing reviewers to share the defects they find over the Internet. The Leap toolkit's flexible process definition mechanism allows it to support the full spectrum of review. It supports the informal walkthroughs as well as the formal inspections. By sharing the defects found during review, the Leap toolkit supports many different types of process improvement. The author can combine the defects from the review with the defects they found during development and gain valuable insight into their development process. The reviewers can see what types of defects they tend to find and what types of defects they miss. They can learn to be better reviewers. By combining the ideas of PSP and FTR the Leap toolkit provides the developer and reviewers with more insight and opportunities to improve.

Whenever data is collected about a process, as in the PSP and FTR, the question of what do you do with this data arises. The use of measurement data raises the issue of Measurement Dysfunction.

## **2.4 Measurement Dysfunction**

Robert Austin introduces the term "Measurement Dysfunction" in his book "Measuring and Managing Performance in Organizations"[2]. He defines dysfunction as "the actions leading to

it fulfill the letter but not the spirit of stated intentions.” In measurement dysfunction, people try, consciously or unconsciously, to change a measure used for evaluation, without trying to change the actual underlying behavior or result that is being measured. The fundamental problem with measurement is that it is impossible to fully measure a behavior or activity. So when people focus on the letter of the measurement they may ignore an important part of the behavior, thus reducing their overall effectiveness.

Austin cites an apocryphal example of measurement dysfunction, a Soviet boot factory. The boot factory was evaluated by the number of boots produced. To meet their quota of boots the factory managers produced only left boots, size 7, since by producing only left boots in one size they could maximize the total output of the factory. Austin uses a study of an employment office by Peter Blau in 1963[3] to provide more insight into measurement dysfunction. The goal of the employment office was to find jobs for their unemployed clients. The employment office employees were evaluated primarily by the number of interviews conducted. The employees responded by focusing as much time as possible on doing interviews, and very little time in finding jobs for their clients. This behavior resulted in client receiving fewer job referrals. When the management changed the evaluation measure to include eight different indicators the employees changed their behavior to improve their standing against various indicators. Some employees destroyed records of interviews that did not result in job referrals and made referrals for clients that did not match the job. In both these situations the true performance of the organizations declined while the measured performance increased.

Austin divides measurements into two categories *motivational measurements*, which are used to affect the people who are being measured, and *informational measurements*, which are used for their logistical, status, and research information they convey. Motivational measurements may lead to measurement dysfunction since the people affected will focus on those measures and change their behavior. A problem with individual measures is they may be used for both motivation and information. Once a measure is taken and recorded managers can use it for status purposes or for evaluation.

#### **2.4.1 Measurement Dysfunction in the PSP**

The data collected in the PSP provides valuable insight into the developer’s development process. The developer learns their development rate, the types of defects they make most often, their average direct hours of work per day, and many other statistics that management could use to evaluate their performance. If management uses this data to evaluate their employees the employees

may start to change their behavior to improve their measures. For example if management says developers should produce 50 LOC per hour and the developer is only producing 40 LOC per hour, they might stop optimizing their code since it takes time and reduces the number of lines of code.

PSP addresses issues by saying that all data is private. difficult to “fake” PSP data when asked to supply data to management. Have to keep two different sets of paper documents.

## **2.4.2 Measurement Dysfunction in Review**

Most training materials for software review indicate that review measures should be used for informational, not motivational purposes. For example, in Tom Gilb’s “Software Inspection” (Addison-Wesley, 1993), he states that one organizational cause of Inspection failure is the “use [of] defect metrics for individual personnel appraisal.” However, once a measure is taken the organization may use that measure in the future for evaluation. As long as the measure is available to management then the potential for evaluation based upon that measure exists. If reviewers turn in their review metrics to management, they may act as if it will be used to evaluate them.

Dr. Johnson wrote a report describing different forms of measurement dysfunction possible in FTR[23]. Those forms are:

- Defect severity inflation. This form of measurement dysfunction can happen when the organization wants to increase the number of important defects found during review. Reviewers can artificially increase the number of important defects by inflating the severity of the defect found during review. Almost every defect becomes critical to the success of the project.
- Defect severity deflation. This measurement dysfunction is the opposite of the above dysfunction. If the goal of the organization is to decrease the number of non-minor defects surviving past a certain phase. In this case the reviewers may reduce the severity of the found defects or reclassify the defects as enhancements.
- Defect density inflation. In defect density inflation, the goal of the organization may be to improve review defect detection as measured by defects detected per volume of work product. In this case, reviewers can improve this metric by classifying defects that would not be counted as being real defects. This increases the number of defects found.
- Artificial defect closure. When the organizational goal is to improve defect closure, reviewers may classify difficult-to-fix defects as “enhancements” and not defects. This reduces the average time to fix defects since the costly ones are no longer defects.

- Reviewer preparation time inflation. Reviewer preparation has been shown to be a key indicator of overall review quality. If the goal of the organization is to increase reviewer preparation, the reviewer may feel pressure to report good numbers. They may inflate the amount of time they prepared for the review.
- Defect discovery rate inflation. Defect discovery rate measures the “efficiency” of the review. It is the average amount of time spent during the review to discover one defect. If the organization wants to increase the discovery rate, the reviewers could either increase the number of defects found or reduce the amount of time they report for preparation.
- Review participation/coverage inflation. The organization may want to increase the participation in reviews or the number/coverage of reviews conducted. In this case there is pressure to conduct many reviews while minimizing the amount of time spent on each review, thus reducing the reviews quality.

Dr. Johnson suggests some possible solutions to the measurement dysfunction issues they are:

- Personal review data should be private. The review data from an individual reviewer should be private so they cannot be evaluated based upon their review data.
- Personal data is always reported in aggregate and anonymously. This allows management access to the data without attaching any individual’s identity to the data. Individuals cannot be evaluated based upon this data.
- Individual process improvement is an individual matter. The measures collected about review should help improve the reviewers and the review practice. The PSP is a strong method for individual process improvement.
- Development group review data is private. Just like the individual, the group may be evaluated on their review data. The group’s data should be protected just like the individual’s data.
- Group review data should always be reported qualitatively. This reduces the chance that the data will be used to evaluate the group.
- Group process improvement is a group matter. Just like for the individual, the group should control the measures they use for improvement.

### **2.4.3 Measurement Dysfunction and LEAP**

Whenever personal data is collected about people, the issue of measurement dysfunction should be addressed. The Leap toolkit cannot stop measurement dysfunction from occurring, but it tries to solve it in two ways. First, all the data is controlled by the user. They can decide what data is shared and what data is kept private. Second, the user has complete control over their data and the Leap toolkit makes it easy to edit the data. This makes measurement dysfunction more conscious. The user knows when they change the data and are more conscious of doing it.

Based upon the PSP, the evaluations of the PSP, our research into formal technical review and the issue of measurement dysfunction, I designed and built the Leap toolkit. The next chapter explains how I built upon all the above work to produce the Leap Toolkit. The Leap toolkit is my attempt to effectively support personal process improvement.

# Chapter 3

## Supporting Software Developer Improvement with LEAP

*Measures of productivity do not lead to improvement in productivity.* - W. Edwards Deming

This chapter discusses Project LEAP and the Leap toolkit. It starts with a brief summary of why I started work on Project LEAP. Then it discusses the design criteria for LEAP-compliant tools. It next discusses the Leap toolkit, which is a reference implementation of the LEAP design philosophy. Next, to demonstrate how the Leap toolkit is used, the chapter presents a usage scenario similar to the PSP example in Section 2.1.3.1. The chapter concludes by discussing three of the intended benefits of the Leap toolkit.

### 3.1 Background

After using the PSP for over two years, I came to appreciate the insights and benefits that it gave me. For example, I learned what my most expensive type of defect was and which defects I could safely ignore until later development. By using the PSP on several Java programs, I was able to accurately predict, plan, and schedule my Java development. I gained a newfound sense of “control” over my software development. I also felt my designs were improving, since I was consciously designing my projects before I started coding them. However, I noticed some general problems with the PSP.

First, I started to question the quality of the data I recorded. I noticed that I did not record all my defects, mostly because the overhead of recording each defect was too expensive.

I disliked stopping my “real” work in order to find the Defect Recording Log, find a pencil, and record the defect. I found that I was just guessing how long it took to fix the defect since I did not have a stopwatch to record the time. I was not recording all the time I was spending developing my programs. Sometimes I would not record time spent designing because I did not have my time recording log available. During this same period of time, Anne Disney and Philip Johnson were conducting a study to look at the data quality of the PSP data. They found that there are significant data quality issues with the manual PSP.[4, 5] This study supported my own feelings that there should be automated support for data collection and analysis in the PSP.

Second, my experiences with industrial partners, management practices, and Robert Austin’s book “Measuring and Managing Performance in Organizations”[2] made me think about the issues of measurement dysfunction in the PSP and review data. Essentially, measurement dysfunction means that an organization’s goals to improve software development and formal technical review may inadvertently pressure their members to produce “good” metrics while actually reducing performance. There are many ways that members can manipulate their personal data collected in the PSP or FTR to get the results that management wants to see. These “right” results do not reflect reality, and software development insights based upon these results could be wrong and actually harm the software development or FTR process.

Third, after four years of PSP usage in academic and industrial settings, I found that the research results on adoption of the PSP are mixed. Pat Ferguson and others report excellent results with adoption of the PSP at Advanced Information Services, Motorola and Union Switch and Signal[9]. However, Barry Shostak and others report poor adoption of the PSP in industry[35, 7]. No research has been published that studies the “long term” adoption of the PSP — i.e., whether or not users trained in the PSP are continuing to use it six months, a year, or more after the training.

Fourth, I felt the lack of group review in the PSP was a major drawback. The most valuable feedback about my software has come from the group reviews our research group conducted on my work. Incorporating other reviewers’ defect data into the PSP data and analysis would provide valuable insights and greatly improve the quality of the programs. Just adding review to the PSP would help, but collecting and managing all the defect data from the reviews would be very difficult. Automated support would make this easy.

These four general problems started me thinking about other software process improvement initiatives. I noticed that many of them suffer from one or more of the following problems:

- Heavyweight development process constraints. Some improvement initiatives impose process constraints on the developers, such as following a strict waterfall development process.

The developers are told that they must follow the development process in order to see any process improvement benefits. Often this development process is generic and does not fit the developer's individual situation. Also many of the improvement initiatives add large overhead to the developer's process. The developer must spend additional effort to collect data and produce reports about the development process.

- **Measurement dysfunction.** Collecting data about the software process such as code size, programmer productivity, defect rate, or estimation accuracy, can often lead to measurement dysfunction. If the developer thinks they might be evaluated based upon these measures they will tend to report the numbers that management wants to see.
- **Organization-level analysis and improvement.** Most traditional software process improvement efforts focus on the organization. The benefits of these methods thus accrue to the organization. Individual developers may indirectly see the benefits of the organization's improvement, but providing benefit to individual developers is not the focus of the improvement effort.
- **Manual data gathering.** Most of the process improvement methods use manual forms. This manual data collection and reporting significantly increases the developers workload. Process improvement activities may require them to interrupt their work regularly to fill out a form. The collation and analysis of these logs is also more difficult because of manual data collection.

These issues motivated me to begin designing an automated, empirically based, personal process improvement tool. My goals were to reduce the collection and analysis overhead for the developer, reduce the measurement dysfunction of the collection process, support many different development styles, and support group review. A tool that meets these goals would provide benefits to the developer and increase long term adoption of empirically based process improvement. To pursue this work, I initiated Project LEAP, <<http://csdl.ics.hawaii.edu/Research/LEAP/LEAP.html>>, and began developing the Leap Toolkit, <<http://csdl.ics.hawaii.edu/Tools/LEAP/LEAP.html>>.



## 3.2 LEAP Design criteria

As part of my initial research, I hypothesized that improved support for software developer improvement would be obtained by attempting to satisfy four major design criteria: light-weight, empirical, anti-measurement dysfunction, and portable.

### 3.2.1 Criteria #1: Light-Weight

The first criteria is that any tool or process I developed for use in software developer improvement should be light-weight. This means that neither tools nor process should create substantial new work for the developer. Data collection should be easy to perform and should not add significant effort to the process. Ultimately, data collection should be automatic and require no conscious effort from the developer. The developer should not have to think about collecting data or be interrupted to collect the data while they are working.

The processes used should not impose a burden on the developer. I do not want the developer to worry about the improvement effort while they are doing software development. They should be focussed on the product not the improvement process. The processes imposed by the tool should be flexible so that they can be used in many different environments. The analyses and other work, like planning and postmortem, should also require as little effort by the developer as possible. The benefit of using the improvement processes should outweigh its cost to the developer. Finally, The developer should see the benefits of the improvement effort as soon as possible after beginning to use the tool.

This criteria implies that any improvement process must be automated as much as possible. A manual process requires too much overhead by the developer. The overhead of recording information by hand and manually doing the analyses will outweigh the benefits of the process.

### 3.2.2 Criteria #2: Empirical

The second criteria is that my methods for software developer improvement should be empirical in nature. I have found the empirical aspects of the PSP and FTR methods to be very useful. The improvements in the development process should be based upon the developer's experiences. I want the developer to use the *observe, evaluate, modify* method for improvement. Each modification is then tested by further observation to see if the change is actually an improvement or is just a false start. By looking at their development empirically the developer can judge for themselves what is best.

### **3.2.3 Criteria #3: Anti-measurement Dysfunction**

The third criteria that my methods must address is the issue of measurement dysfunction. Data collected for the improvement effort could be misused. If there is measurement dysfunction, then the data collected and analyses will not reflect reality. Any insights gained from this data and analyses will be faulty and may cause more problems than they solve. This issue is important since the software development process is very interesting to people other than the developer. Solving measurement dysfunction may be impossible, but the issue should be recognized and its impact minimized as much as possible.

### **3.2.4 Criteria #4: Portable**

The fourth criteria is that any tool or process used in software development improvement should be portable. In the Internet age, software developers often change jobs or positions and the tool support for their development improvement should be portable. To continue their improvements when they change jobs or organizations, the developers should be able to take their data and the tool support with them. A tool that supports developer improvement that cannot follow the developer as they move is not going to help that developer for very long. The tools, and data must be able to run on many different platforms and be able to support many different development styles, languages, and processes.

## **3.3 Leap toolkit: a reference implementation of the LEAP philosophy**

I developed the Leap toolkit as a reference implementation of the four LEAP design criteria. The Leap toolkit explores some of the practical issues behind the design criteria.

### **3.3.1 Supporting personal process improvement**

The Leap toolkit is based strongly upon the PSP. The Leap toolkit uses the three primary data types, defects, size, and time, from the PSP. However, unlike the PSP, developers are able to choose what types of data to collect to help them meet their process improvement goals. If the developer is just interested in improving their estimation ability, they can record the size of their projects and the amount of time it takes them to complete them. The Leap toolkit provides the developer with different time estimation tools.

If the developer wants to prevent defects, then they can just record their defects and not worry about size or time. The Leap toolkit can analyze their defect data and provide them insight into which defects occur most often and the developer can generate checklists that help them find those defects.

The Leap toolkit does not impose a development process on the developer. They can use their own process. The Leap toolkit supports many different process definitions.

### **3.3.2 Supporting Group Review**

From FTR, I took the idea of supporting multiple developers reviewing a work product and sharing the defects they find. The defects that others find in your work product may be more important than any of the defects you find in your own work product. By incorporating support for sharing defect data, the Leap toolkit can support reviews. In the Planning phase, review leaders can define the work product, project, defect types and checklists for the reviewers. During the preparation phase, the reviewers can use the Leap toolkit to record the defects they find in the work product. They can send their defects to the review leader who can use the Leap toolkit to combine the defects into a single list. During the review meeting the review leader can display the combined defects and each may be discussed. The author of the work product can take the combined list of defects and add it to any defects that they found. This provides the author with more data about their development process.

The Leap toolkit's flexibility allows the review leader to define their own process, defect types, and decide what review metrics they are interested in recording. The Leap toolkit also allows each reviewer to record their effort and the defects they find. The Leap toolkit can analyze the defect, time, and size data to produce reports on the defect density, defect detection rate and effectiveness of the review process.

Individual developers can easily start a group review using the Leap toolkit. They begin by distributing their work product and a special Leap data file containing review definitions. The reviewers use the Leap toolkit to record the defects they find and then email or send their results back to the developer. The developer can easily merge all of the individual defect data into the Leap toolkit and use the data to improve the work product and their development process.

### **3.3.3 Providing Light-Weight Support**

The Leap toolkit tries to reduce the overhead of software developer improvement by automating many data collection processes and analysis calculations and conversions. The Leap toolkit provides tools to record time and defect data. These tools reduce the developer's overhead. The developer just has to click a few buttons and enter a description to collect time and defect data.

The Leap toolkit, unlike the PSP or PSP Studio, does not impose a specific development process on the developer. If the developer wants to use the same process as PSP2.1, they may. If user does not want to have a design phase, Leap will also support that process. The user can define their own processes and Leap will support data collection and analyses based upon their processes.

The Leap toolkit also allows the user to define their own size types. This allows the user to choose a size measure that is more effective and/or convenient than lines of code used in the PSP.

### **3.3.4 Supporting Empirical Data Analysis**

The Leap toolkit allows the developer to record their effort, work product size and the defects they make while developing software. Based upon historical projects the Leap toolkit helps the developer to produce an estimate for the total amount of effort the next project will take.

The data analysis features allow the developer to evaluate how effective their development process is. They can easily see how accurate their plans were, what the most expensive defect was, and where they spent the most time. Based upon the data they collect and analyze, the developer can evaluate changes to their development practice.

### **3.3.5 Reducing Measurement Dysfunction**

No tool can stop measurement dysfunction. My philosophy in the design of the Leap toolkit is to acknowledge that measurement dysfunction can occur in both personal software process improvement and review. To address these issues I allow the developer full control over the data shared. The developer can decide exactly what data is shared and edit the data. This moves any measurement dysfunction issues from the background to the foreground.

The Leap toolkit stores all its data in ASCII files. This allows the developer to control the access to the data. Also, the Leap toolkit gives developers complete control over the data that they share. Developers have full control of where they save their Leap data. When users use the Leap toolkit's email capability to share their data, the Leap toolkit asks the user what data to send. No data is shared without the user's knowledge.

The toolkit also makes it very easy to edit the data before it is sent or saved. We did this for two reasons. First, if there is a data collection error then the user can edit the data to correct the error. Second, the user can edit their data before they provide it to another person. This allows the user to decide what data other people see.

Even though no tool can stop measurement dysfunction, an improperly designed tool can create measurement dysfunction. If the user feels they have no control over their data, they may feel pressure to provide the “right” data and modify their behavior accordingly. This lack of control may encourage measurement dysfunction.

### **3.3.6 Providing a Portable Tool**

Since the Leap toolkit is written in Java, it can run on many different computer platforms. By using ASCII files for data storage users can easily put the files on a disk or transfer them. Both of these features allow the user to take their data and the Leap toolkit with them when they move.

A Leap toolkit user showed me how important this feature is. They started using the Leap toolkit on a Windows NT machine. They recorded their development process and common defects. After about a year they changed jobs and took their Leap data with them. In their new job they were working on a UNIX workstation running Solaris. They downloaded the Leap toolkit for Solaris and were able to keep using the Leap toolkit.

## **3.4 The Leap toolkit’s architecture**

The Leap toolkit architecture has evolved over the last two and a half years. I started development of the Leap toolkit in the summer of 1997. I chose to implement the Leap toolkit in Java since Java programs run on many different platforms.

The development of the Leap toolkit followed some of the ideas of action research [13]. Hult and Lennung define action research as:

Action research simultaneously assists in practical problem-solving and expands scientific knowledge, as well as enhances the competencies of the respective actors, being performed collaboratively in an immediate situation using data feedback in a cyclical process aiming at an increased understanding of a given social situation, primarily applicable for understanding of change processes in social systems and undertaken within a mutually acceptable ethical framework[16].

The problem to be solved is how to improve individual software development while enhancing the competency of the software developer, myself as a researcher and the Leap toolkit.

At first the Leap toolkit supported only defect collection, defect analysis, and group review. After a few months of use in our research group, I added support for size, and time data. By the summer of 1998 I had added checklists and patterns. In late 1998, I used the Leap toolkit in a software engineering course at the University of Hawaii, Manoa. Feedback from the students and members of my research group changed the focus of development to user interface and usability issues. I worked on project estimation and postmortem support. Much of the work over the last two years has explored the interactions and implications of the original four design criteria.

There have been over 30 public releases of the Leap toolkit since the Fall of 1997. As of January 12, 2000, the Leap toolkit consisted of 44,000 lines of code, 2,209 methods, 287 classes, and 14 packages.

### 3.4.1 Data Files

Early versions of the Leap toolkit wrote out data in binary format as serialized Java objects. I chose this simplistic design because it was trivial to implement at a time when quickly obtaining a running prototype for preliminary user evaluation was paramount. However, this design had a number of severe shortcomings, including:

- Java binary format is difficult to generate by non-Java tools. It prevented us from expanding the Leap toolkit to include tools based upon Visual Basic, Emacs, etc.
- The binary data files became invalid as soon as we upgraded our internal tools, such as JClass Livetable.
- Binary data files require encoding for sending via email. Sending Leap data files via email greatly simplifies the problem of conducting a distributed group review. Reviewers can review the work product locally and email their defects to the review leader.
- Binary data files are not human readable. Concerns about exactly what is being saved in the files could lead to measurement dysfunction.

As a result of these problems with binary data files, I redesigned the input-output mechanisms in the Leap toolkit to use an ASCII format. The LEAP toolkit now uses a *restricted* form of the HTML table format to store its data. This design has the following strengths:

- A non-binary format for primary Leap data files is important for portability and minimization of measurement dysfunction. Old data file formats can easily be read in and converted to the

new data file format. Since the data is ASCII text it can be sent via email without changing the format. This simplifies the exchange of Leap data files during group review.

- Use of HTML for formatting allows the file to be human readable and viewed in a browser easily. The user can quickly check their Leap data outside of the Leap toolkit.
- The HTML table environment can collect individual data records together into a dataset.
- The HTML table environment allows me to write parsers for input of data collected from other sources, such as Excel or Word files written as HTML.
- The HTML table environment allows many different tools to write Leap compliant data. Dr. Johnson wrote a very simple Emacs tool to produce Leap defect data files. This tool reduced the overhead of recording defects while using Emacs. The data files could be combined with the data files generated by the Leap toolkit for later analysis.
- The HTML table environment allows Leap tools to store multiple datasets within individual files. This flexibility allows the user to store different types of data in different files. For example, I keep all my common definitions, document types, phases, size definitions, defect types, etc., in one file. I store my raw project data, size, time, and defects in separate data files. Finally, I store my project definitions in yet another file. Conversely, students using the Leap toolkit often keep all their data in a single file.

Figure 3.1 shows a portion of a Leap data file. Each table has a table type, a formatting type and a version. In the figure are two tables each with two entries. The first table (Table Type: DCT, Formatting Type: Plain, and Version 1.0.0) defines two document types, Java Source and Lisp Source. The second table (Table Type: DFT, Formatting Type: Plain, and Version 1.0.0) defines two defect types for Java Source.

As the Leap toolkit evolved, I added new tables. The Leap toolkit's architecture supports the definition and addition of new data tables, by using a generic HTML table parser and a data structure that knows the format of each table. The HTML table parser reads in the HTML data and converts them into a Vector of Strings. The LeapFileReader looks at the type and version of the table that was read and then converts the Strings in to objects that are stored in the Leap data sources. Figure 3.2 shows how the data is read in from the Leap data files to the leap data sources and written out from the Leap data sources to the leap data files. To add a new table type or change the order of the data in a table the Leap developer updates the Leap data structure holding the table

```

<html>
<head>
<title>LEAP Data File</title>
</head>
<body>
<table border>
<tr>
<td>DCT Plain 1.0.0
<tr>
<th>Name<th>Description
<tr>
<td type=Name>Java Source
<td type=Description>Java source code
<tr>
<td type=Name>Lisp Source
<td type=Description>Lisp source code (Emacs or Common)
</table>
<table border>
<tr>
<td>DFT Plain 1.0.0
<tr>
<th>Name<th>Description<th>DocType
<tr>
<td type=Name>l0: Syntax
<td type=Description>General syntax errors
<td type=DocType>Java Source
<tr>
<td type=Name>l1: Misspelled
<td type=Description>Misspelled identifier
<td type=DocType>Java Source
</table>

```

Figure 3.1. A short section of a Leap data file. Notice the HTML tables. Each row in the table is a data entry.



format information. When old data files are read in, they are converted to the newest version by the LeapFileReader.

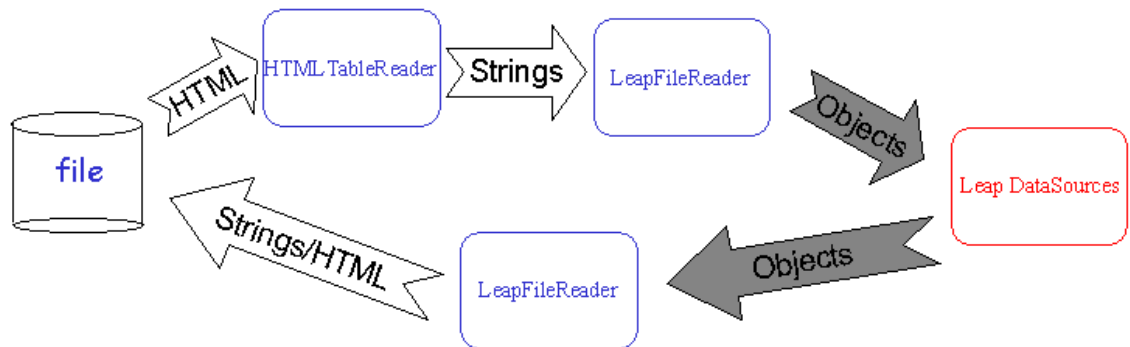


Figure 3.2. Leap File IO architecture.

This data file architecture allows the user to load many files with different data tables in them. The Leap data sources combine all the data into a single data source for each table. When the user saves their data each data entry is saved back to the file where it was loaded from. Figure 3.3 shows an example of the combining of Leap data from different files. The user loads four data files, `prjs.leap`, `defs.leap`, `foo.data`, and `bar.data`. The raw data from `foo.data` and `bar.data` is combined in the Leap data sources. For example, the defects for both projects are combined in the defect data source. When the user saves the data, it is written to the file it was loaded from. The size, time, and defect data for project #1 is written to `foo.data` and the data for project #2 is written to `bar.data`.

The purpose for this file handling architecture is to give the user full control over their data. They can organize their data anyway they like. For example they could store all their data in a single file or split their data in raw data files and definition files or they could store all their data based upon in files organized by date. This mixing of data from different files has caused confusion in Leap toolkit users. They often do not know where the Leap toolkit saves the new data entries.

If I were to redesign the Leap data file system, I would change this model. I think a simpler model that allows only one active data file and importing data from other files might be easier to implement for the developer and learn for the users. Another solution for this problem would be to provide a private database for each developer. All their data would be stored in the database and they would have complete control over the access to the database.

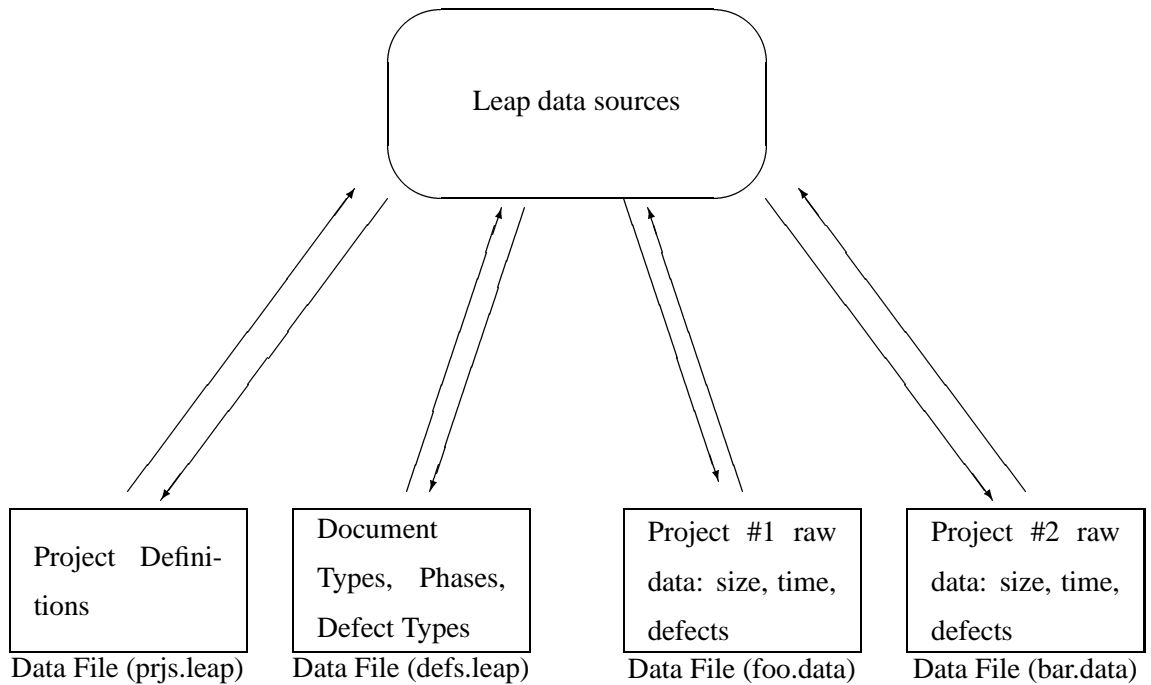


Figure 3.3. Leap data file loading and saving example.

### 3.4.2 Model-View-Controller

Once the data is loaded into the Leap toolkit, the Leap toolkit uses the Model-View-Controller architecture to manage the data in the program. The Model-View-Controller architecture provides great flexibility to the Leap architecture. Each data type, and there are currently thirteen, has its own model and viewer. The data models are the Leap data sources described above. Each data type also has a table based viewer. Figure 3.4 shows the Model-View-Controller architecture and some of the different Leap toolkit objects. There are many controllers that handle interactions between the different data types. Some of the controllers deal with loading, saving data and transferring the Leap data. The Model-View-Controller architecture is somewhat reflected in the Java package structure of the Leap toolkit.

### 3.4.3 Java Packages

I divided up the Leap toolkit into twelve different Java packages. The Java package structure allows me to separate the functionality of Leap toolkit. Figure 3.5 shows how the packages build upon each other. The lowest layer in the Leap toolkit, `inter`, is a set of Interfaces that define the overall Model-View-Controller architecture. These interfaces are used by all the higher pack-

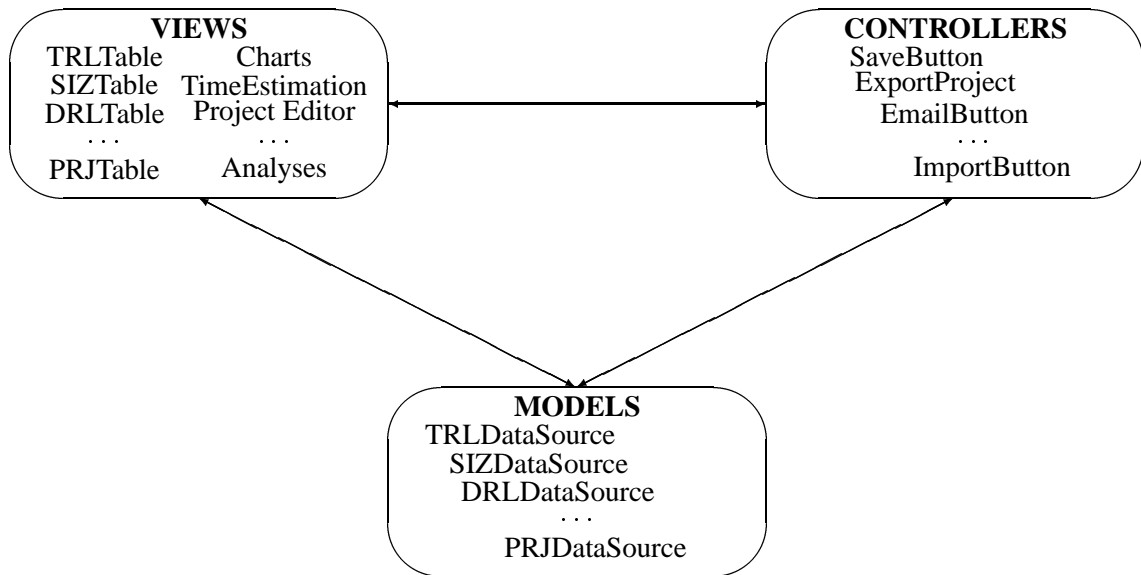


Figure 3.4. Model-View-Controller architecture in the Leap toolkit.

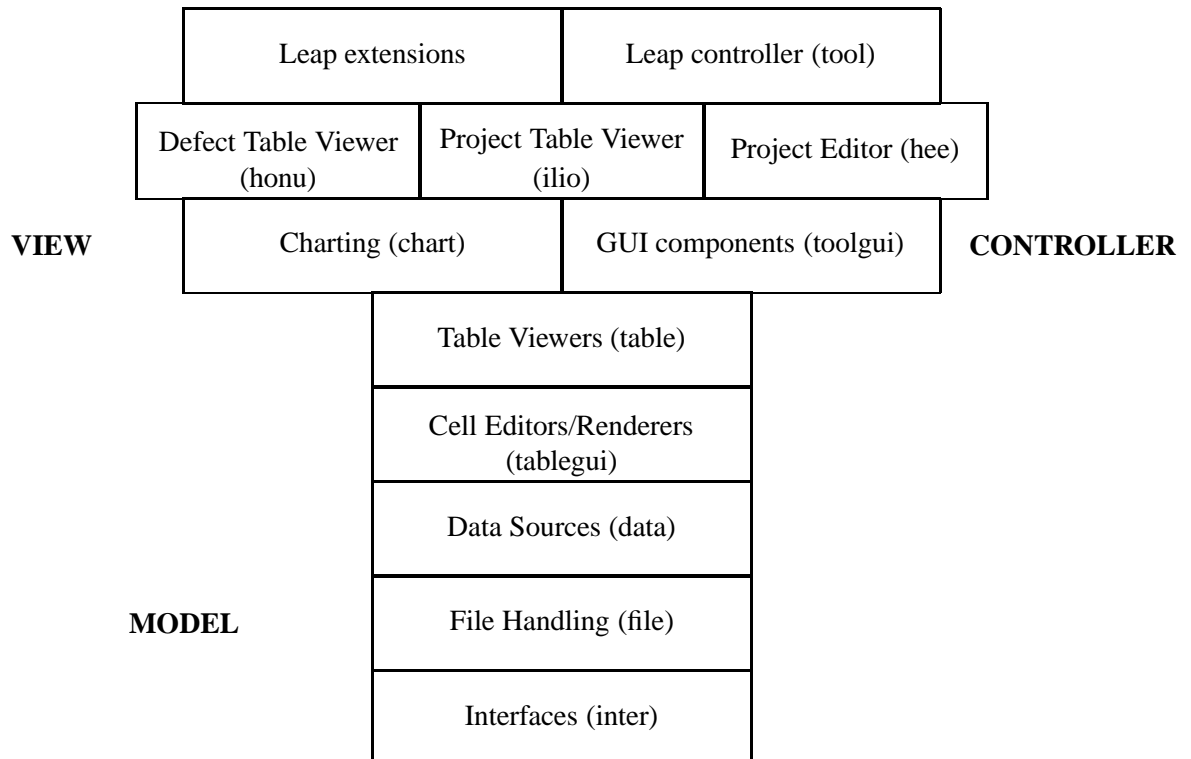


Figure 3.5. Leap package architecture

ages. The model portion of the architecture consists of the `file` and `data` packages. The `file` package handles all the file I/O. It reads in the HTML data files and builds the Java objects stored in the Leap Data Sources. The file layer does not build the Leap data sources it uses the Model interface defined in the `inter` package to pass the object to the third layer. The third layer is the `data` package. The `data` package implements all the Leap Data Sources. The data sources are given the objects from the file layer and build the data sources. The data sources are a central repository for all the Leap data.

The next higher package, `tablegui`, starts the View-Controller portion of the Leap architecture. It contains all the presentation editors and renderers for the table views. This package gives the Leap tables their common look and feel. The last package in the direct hierarchy is the `table` package. This package defines all the Leap tables. They are the primary views on the Leap data sources. Above the `table` package are two packages, the GUI components and controllers (`toolgui`) and the charting package (`chart`). The `toolgui` package defines some GUI components, like buttons, controllers like the email mechanism to sent the Leap data over a network and a generic display for Leap tables. The `chart` package defines some common charting mechanisms based upon the Leap tables. For example, there is a single table summary chart that will build a bar chart of any table based upon two columns in the table. One column defines all the X Axis values the other column is the values for the X Axis. Above the `toolgui` and `chart` packages are more specific viewers like the defect viewer and project viewer. Additionally there is a single project viewer/manager, `hee`, that allows the user to create and view projects. Above the specific viewers are two packages, the `tool` package and the `extension` package. The `tool` package defines the main Leap controller. This class is the one that the user starts to run the Leap toolkit. The Leap toolkit extensions facility is discussed in section 3.4.5.

### 3.4.4 Generic Analyses

The Model-View-Controller architecture allows me to easily implement some generic analyses. The `chart` package provides some generic charting mechanisms that work on tables. For example, depending upon which column and table is used for the data source, the Leap toolkit can produce different charts. Figure 3.6 and 3.7 use the same charting mechanism but use different columns of the defect table. This is a very powerful tool that allows me to add new analyses quickly.

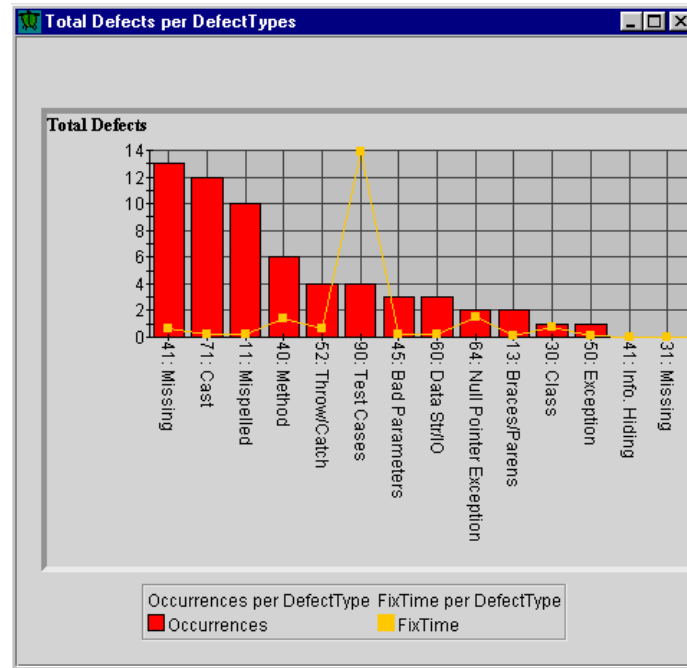


Figure 3.6. Defect analysis, number of defects and the total amount of time it took to fix them per defect type.

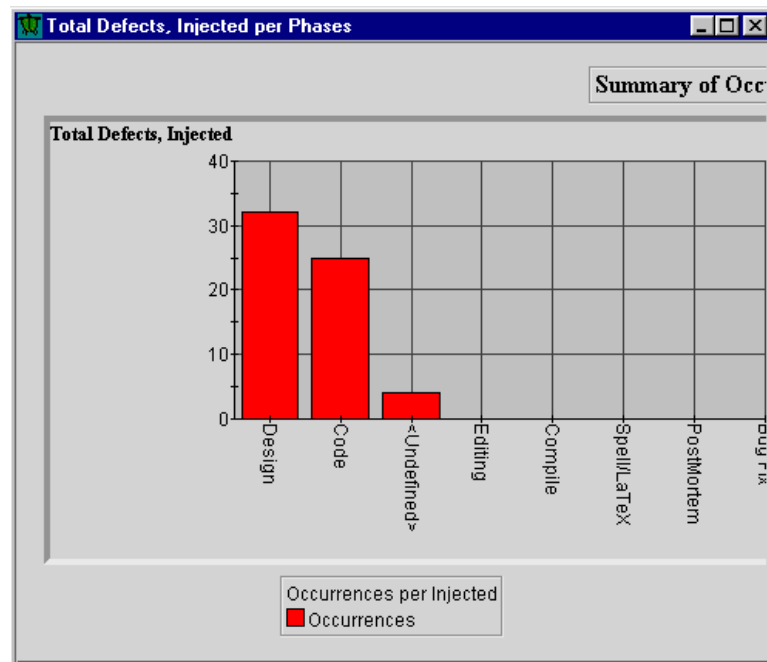


Figure 3.7. Defect analysis, number of defects injected per phase.

### 3.4.5 Extension mechanism

The Java language's reflection mechanism allows Java programs to obtain the fields, methods and constructors for a given class name. This mechanism allows the Leap toolkit to have a very flexible extension mechanism.

Software developers can build their own classes that are loaded into the Leap toolkit. To be a Leap toolkit extension, the developer's class must implement the LeapToolExtension interface. The LeapToolExtension has two methods, initialize and getName. When the developer is done building their extension they place their extension's class files in the Leap toolkit extensions directory.

When the Leap user chooses the Extensions menu on the Leap toolkit controller, the Leap toolkit searches the extensions directory and uses reflection to find all the classes that implement the LeapToolExtension interface. The toolkit builds a dialog box that allows the user to choose which extensions to load. Figure 3.8 shows the dialog box with only one extension: the Single Line Defect entry tool, Mano. When the user chooses the extension to load, the Leap toolkit calls the extensions

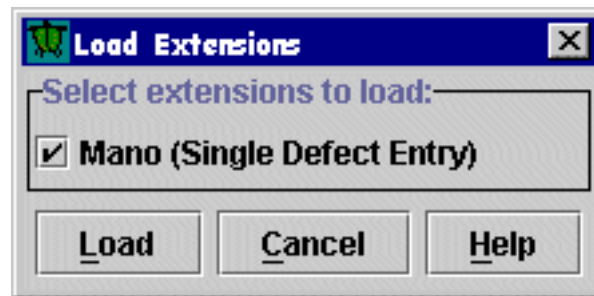


Figure 3.8. The Leap toolkit extension dialog. The Leap toolkit found only one extension in its extension directory.

initialize method with itself as the parameter. The extension then initializes itself and has full access to the Leap toolkit.

The next section describes the use of the Leap toolkit.

## 3.5 An Example use of the Leap toolkit

This example is similar to the PSP example in section 2.1.3.1. The project, developer, and environment are the same, but this time Cam will use the Leap toolkit instead of the PSP.

**Starting a new Project** Cam is called into Philip's office to get the requirements for the next project. "We need to extend our existing Palm calendar application by adding multiple user support." Cam and Philip go over the requirements for the new Multi-user features. By the end of the meeting Cam understands the requirements for the new features. Philip asks Cam how long the project will take. Cam replies that he needs to analyze the requirements a bit more and will give Philip an answer in a few days.

**Design** Cam returns to his cubicle and starts the Leap toolkit with all his historical Leap data. Figure 3.9 shows the Leap Controller. He then opens up the Projects viewer Ilio and right mouse

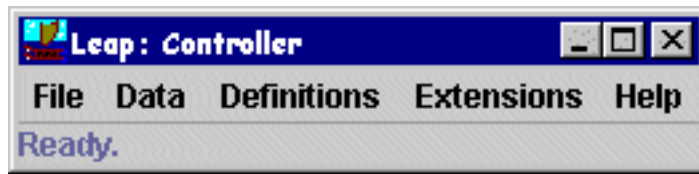


Figure 3.9. Leap Toolkit Controller. This is the main controller for the Leap toolkit.

clicks on a blank line to start Hee the project editor. He fills out the name, description, and start date for the Multi-user Calendar extension project. He then decides what PhaseSet to use during development. The PhaseSet is the set of Phases Cam is going to use. Cam has already defined a PhaseSet called "Development" that consists of the following phases: Design, Planning, Code, Compile, Test, Release, BugFix and PostMortem. Figure 3.10 shows the Hee project editor after Cam has entered this data. Cam closes Hee and saves the project data.

Next, Cam starts his first phase of development, Design. He starts the single line timer tool Io, chooses the Multi-user Calendar project, the Design phase and clicks the Start button. Figure 3.11 shows the Io tool recording time for the Design phase.

Cam begins to design the project. He creates the Java Class files with all the methods and constructors. He comments each class and method describing what the class and method does, what parameters it needs and the return values. While he is designing the classes and methods he gets a phone call. To record the interruption he hits the pause button on Io and takes the call. Figure 3.12 shows Io paused while Cam answers the phone. When he is finished with the call, he un-pauses Io and continues designing.

**Planning** Once he finishes designing all the classes and methods, he runs LOCC over the source files to count the number of packages, classes and methods he has designed. Cam stops Io and

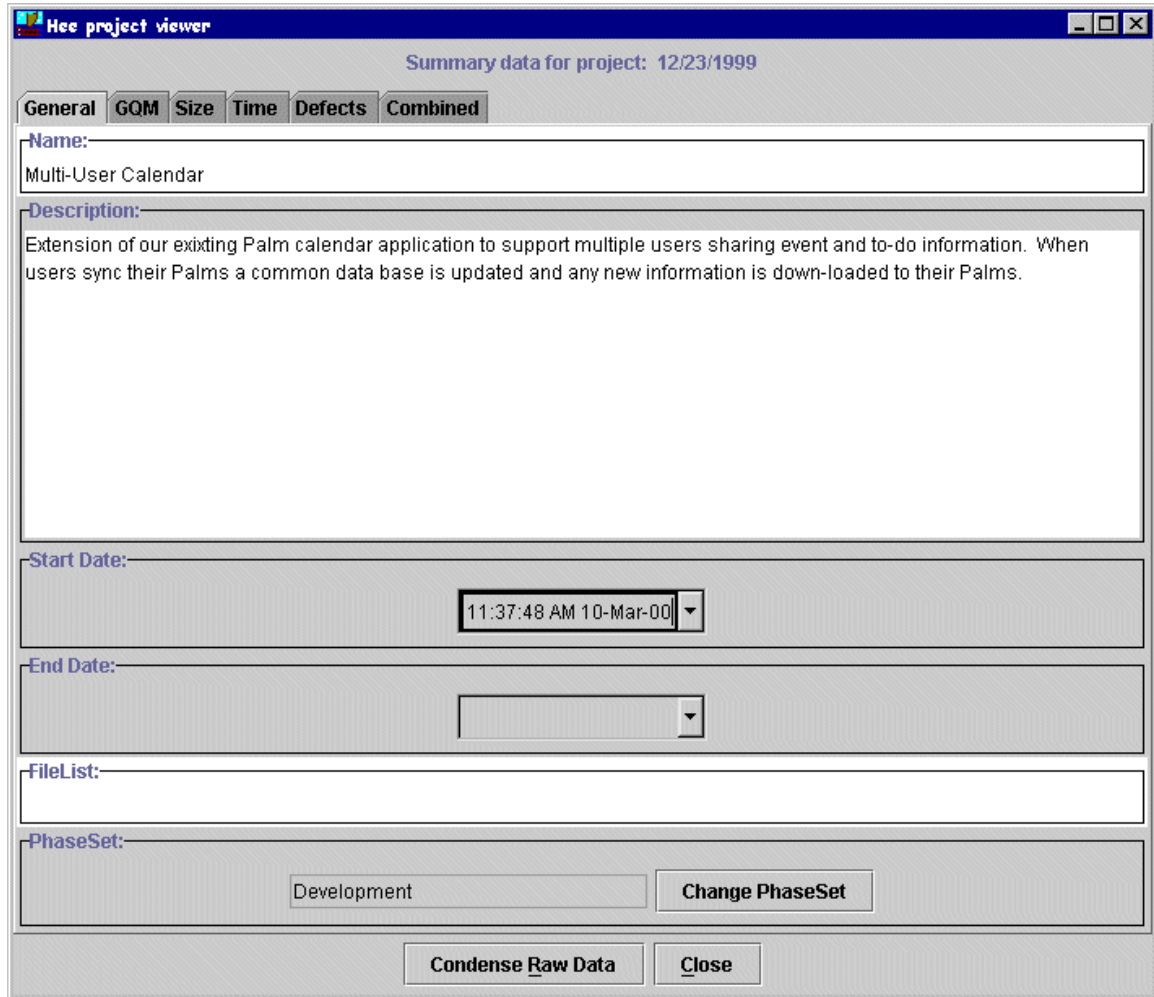


Figure 3.10. Hee, the project editor. Cam has defined the Multi-User Calendar extension project and chosen his Development process.



Figure 3.11. Io time recording tool. Cam has just started recording time for the Design phase of the Multi-User Calendar extension project.



Figure 3.12. Io time recording tool. Cam has just interrupted recording time to answer the phone.



records his design time. He then changes phases to planning and starts the timer again. To help



Figure 3.13. Io time recording tool. Cam has just started his planning phase.

him estimate the number of LOC needed for the project Cam opens the Project Comparison tool in Leap. Figure 3.14 shows the Project Comparison tool with all Cam's historical data. From this

Project Comparisons				
Size	Time	Defects	Combined	
Leap Extensions	Merge Rows	Single Line Defect	Single Project Viewer	Total
JavaSize	JavaSize	JavaSize	JavaSize	JavaSize
231 LOC	217 LOC	630 LOC	403 LOC	7,822 LOC
6 Method	7 Method	19 Method	68 Method	533 Method
3 Class	3 Class	1 Class	13 Class	75 Class
2 Package	1 Package	0 Package	1 Package	10 Package
8.500 LOC/Method	31.000 LOC/Method	33.158 LOC/Method	5.926 LOC/Method	14.863 LOC/Method
7.000 LOC/Class	72.333 LOC/Class	630.000 LOC/Class	31.000 LOC/Class	105.627 LOC/Class
5.500 LOC/Package	217.000 LOC/Package	∞ LOC/Package	403.000 LOC/Package	792.200 LOC/Package
2.000 Method/Class	2.333 Method/Class	19.000 Method/Class	5.231 Method/Class	7.107 Method/Class
3.000 Method/Package	7.000 Method/Package	∞ Method/Package	68.000 Method/Package	53.300 Method/Package
1.500 Class/Package	3.000 Class/Package	∞ Class/Package	13.000 Class/Package	7.500 Class/Package

Figure 3.14. Project Comparison tool. Cam is comparing all his Java development projects to see his average LOC/Method.

comparison he sees that on average his methods have 15 LOC. Given his initial design of 5 classes and 80 methods he estimates that the project will have 1250 LOC. He adds an extra 50 LOC since he feels that the project is slightly more complex than the average project.

Cam reopens Hee and goes to the Size tab to enter in his planned size. Figure 3.15 shows Cam's planned size for the Multi-user Calendar project.

Hee project viewer

Summary data for project: Multi-User Calendar 12/23/1999

General GQM Size Time Defects Combined

JavaSize ▼ New Size Definition

Level	Planned	Actual	Error	%
LOC	1,250.0	0.0	-1,250.0	-100.0
Method	80.0	0.0	-80.0	-100.0
Class	5.0	0.0	-5.0	-100.0
Package		0.0	0.0	0.0

Condense Raw Data Close

Figure 3.15. Hee Size tab. Cam has entered in his planned size for the Multi-user Calendar project.

He then goes to the Time tab to estimate the amount of time the project will take. The Leap toolkit provides a Time Estimation tool that Cam starts. Figure 3.16 shows the Leap Time Estimation tool. Cam is able to view his historical planned size vs. actual time or his historical

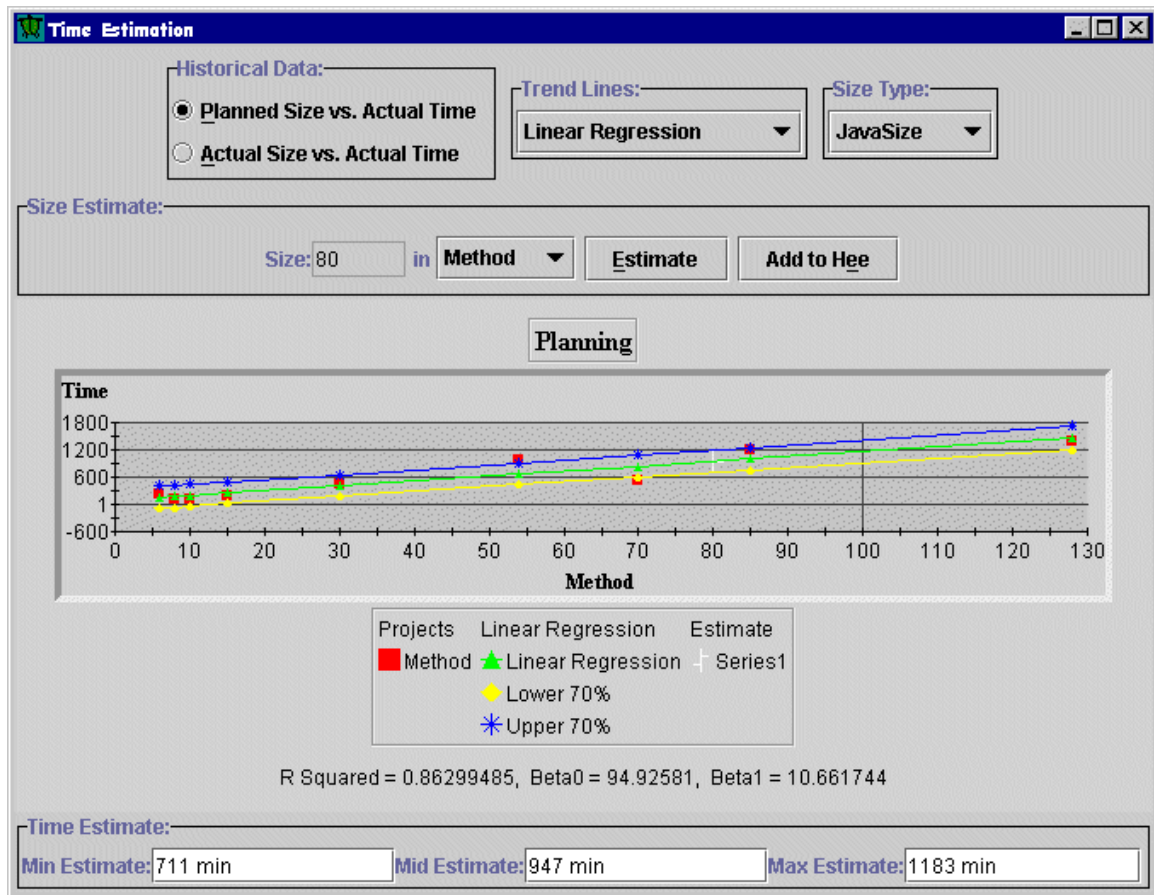


Figure 3.16. Time Estimation Tool. Cam has chosen to estimate based upon his planned sizes, linear regression model, and methods.

actual size vs. actual time. He can also choose from five different trend line models to help him estimate the amount of time the project will take. In Figure 3.16, Cam is using a Linear Regression trend line since the  $r^2$  value is high, 0.86. The Time estimation tool automatically takes Cam's estimated sizes and uses them as the size estimate. When he presses the Estimate button the tool fills in the Min, Mid and Max estimated time automatically. In this example the estimate has a range from 711 minutes to 1183, with a best estimate of 947 minutes. Cam presses the Add Estimate to Hee button and the estimate is added to a table on the Time tab in Hee. Figure 3.17 shows the Time Tab in Hee with several different time estimates based upon different combinations of trend lines and size categories. Generating different size estimations takes only a few seconds, so Cam is able

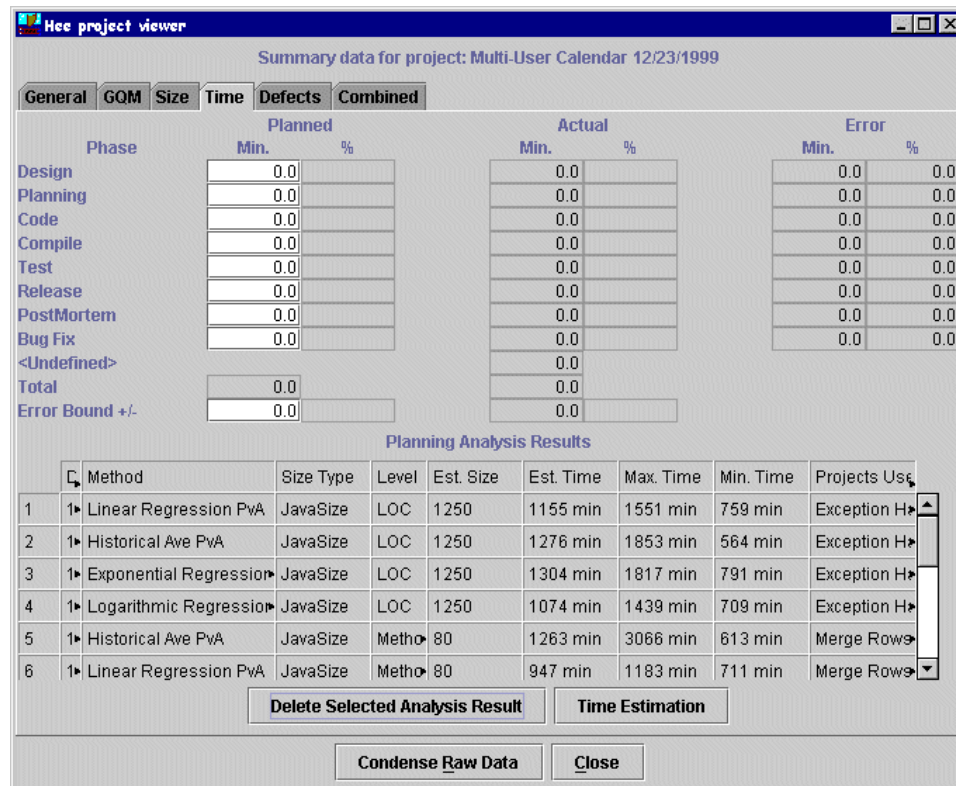


Figure 3.17. Hee Time Tab. Cam has added six different time estimations based upon different trend lines and size categories.

to compare the different results quickly. Based upon the six time estimates, Cam decides that the project should take 1200 minutes. To distribute this time among the different phases, Cam again starts the Project Comparison tool and looks at the Time comparison. Based upon his historical

Project Comparisons

SizeTimeDefectsCombined

Single Project Viewer				Totals	
0.000	Development	1	0.186	Development	10.019
5.528	Planning	27	5.009	Planning	1172.256
7.538	Design	209	38.776	Design	98018.897
25.126	Code	301	55.844	Code	2,00238.604
27.136	Compile	1	0.186	Compile	4208.099
31.156	Test	0	0.000	Test	1,43027.574
0.503	Release	0	0.000	Release	881.697
0.000	Bug Fix	0	0.000	Bug Fix	681.311
3.015	PostMortem	0	0.000	PostMortem	801.543
	Total	539		Total	5,186

Ok

Figure 3.18. Project Comparison Time Tab. Cam is comparing all his Java development projects to see how his time is distributed between the phases.

time distribution, Cam enters in his planned Times in Hee. He also believes that his estimate will be accurate to within two hours. Figure 3.19 shows Cam's estimated time for the Multi-user Calendar project.

Given his estimated size and estimated time, the Leap toolkit automatically calculates his estimated productivity in the four different size levels, 62.5 LOC/hour or 4 methods/hour or 0.2 classes/hour. Figure 3.20 shows Cam's predicted productivity.

Based upon his historical time data, Cam knows he has three direct hours of work per day for programming tasks. If all goes according to plan Cam will be finished with the Multi-User

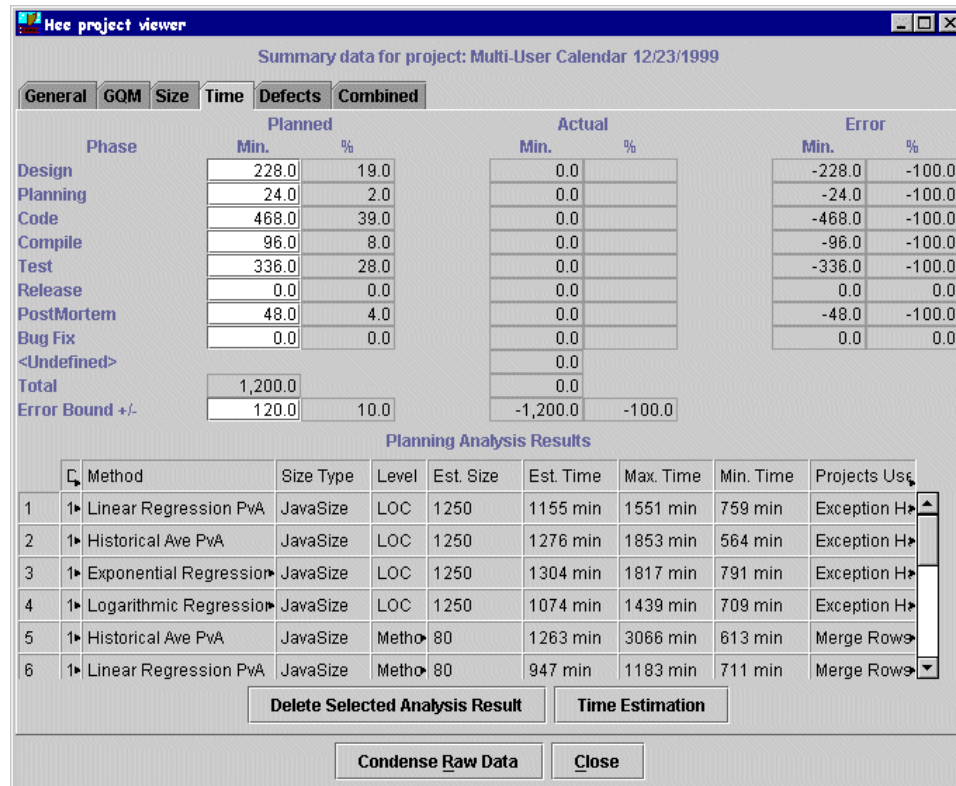


Figure 3.19. Hee Time Tab. Cam has filled in his planned time for the project.

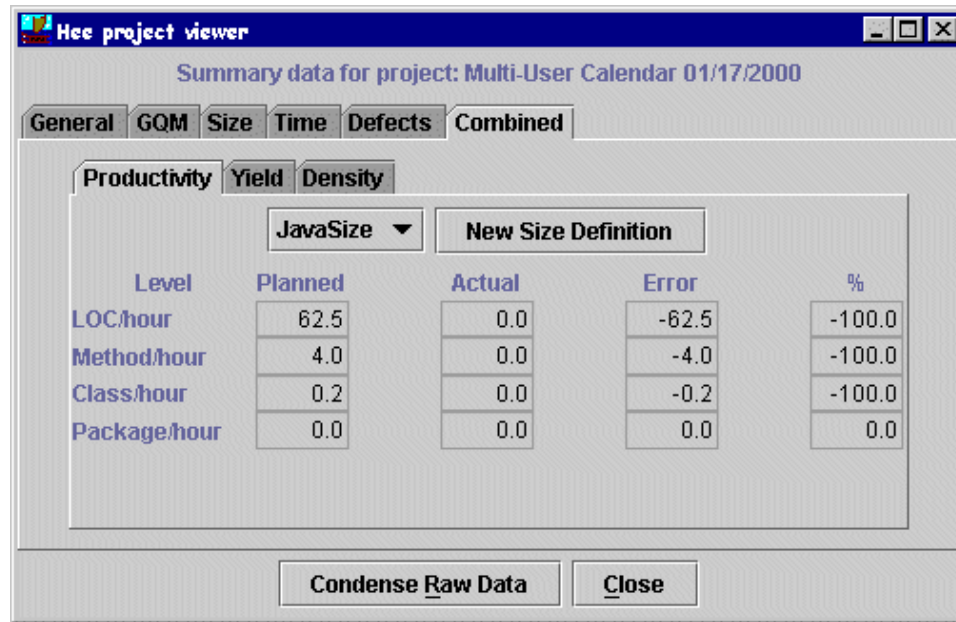


Figure 3.20. Hee Combined Tab. The Leap toolkit has calculated Cam's expected productivity for the Multi-User Calendar project.

calendar project in a week and a half ( $1200/180 = 6.67$  days ). Cam hits the stop button on Io and records his planning time.

Cam does not believe that predicting when he will inject and when he will remove his defects is very useful. He has not gotten much insight into his development process by predicting his defects so he just schedules a meeting with Philip for that afternoon. At the meeting he tells Philip that the Multi-User calendar extension should take between a week (lower prediction interval) and two weeks (upper prediction interval) to complete. Cam's best estimate is that it will take a week and a half. Philip agrees that this is a reasonable schedule for the project.

**Design Review** Now that Cam is finished with his design and planning, he want to have the other developers review his design. Cam produces a JavaDoc design document from the source files. The JavaDoc design document describes all the classes and methods for the Multi-User calendar project. Cam publishes the JavaDoc pages on the development groups web server. Cam send Robert, Joe and Tie an email asking if they will review his design. In the email he tells them the URL to the JavaDoc design documents and an URL to the review's Leap definitions. Cam has created a Leap data file with all the definitions that the reviewers will need for the design review including Cam's defect types, the document type, the Multi-User Calendar project definitions and Cam's severity

levels. Cam also asks the reviewers to send him the results of their reviews by the end of the next day. He also tells the reviewers not to spend more than one hour on the review.

When a reviewer starts to review Cam's design they open their Web browser to the JavaDoc design pages and start the Leap toolkit. Once the Leap toolkit has started they can load Cam's definitions by using the Load URL option (see figure 3.21). When they find a defect in the

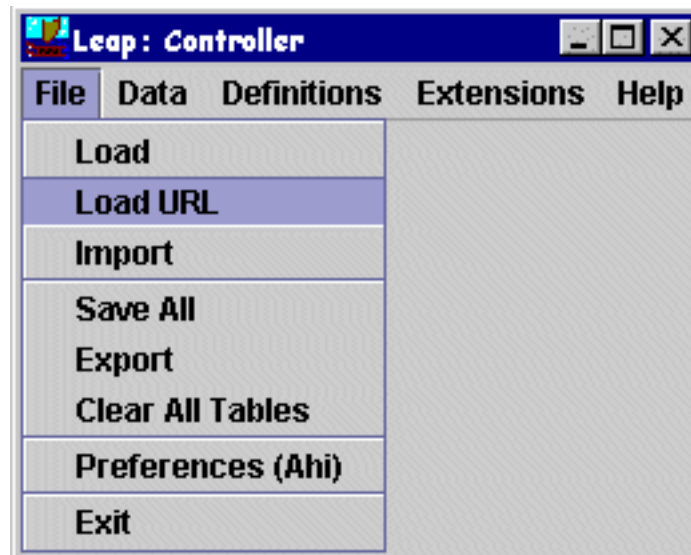


Figure 3.21. Loading a Leap data file from an URL.

design, they record the defect in Leap using the Defect Recording Tool extension (Mano). Figure 3.22 shows Mano recording a design defect. When they are finished with their review, they open

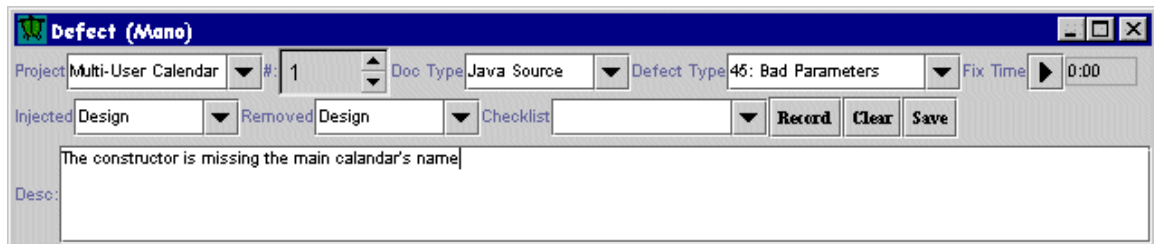
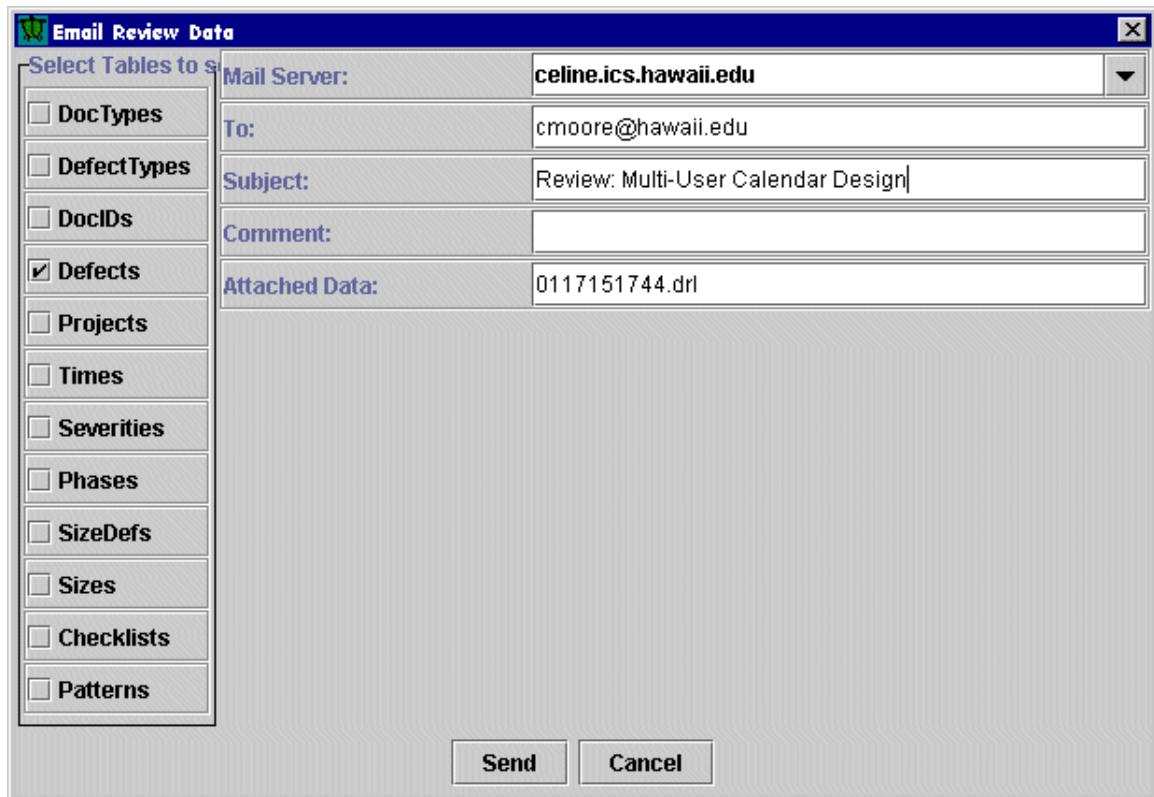


Figure 3.22. Defect Recording Tool Mano. Shows a design defect in the Multi-User Calendar project. The defect is a bad parameter defect where a parameter is missing from a constructor.

Honu, the defect table viewer, to check that all the defects they found are correct. They then hit the Mail button to send Cam their defects. Figure 3.23 shows the Email Review Data window. The reviewer fills in the mail server, Cam's email address, the subject line for the email message, and any comments they have. The Leap toolkit automatically fills in the attached data file name. The





The image shows a software window titled "Email Review Data". On the left is a vertical list of checkboxes under the heading "Select Tables to send". The "Defects" checkbox is checked, while others are not. On the right are input fields for "Mail Server:", "To:", "Subject:", "Comment:", and "Attached Data:". The "Mail Server" field contains "celine.ics.hawaii.edu", "To" contains "cmoore@hawaii.edu", "Subject" contains "Review: Multi-User Calendar Design", and "Attached Data" contains "0117151744.drl". At the bottom right are "Send" and "Cancel" buttons.

Select Tables to send	
<input type="checkbox"/>	DocTypes
<input type="checkbox"/>	DefectTypes
<input type="checkbox"/>	DocIDs
<input checked="" type="checkbox"/>	Defects
<input type="checkbox"/>	Projects
<input type="checkbox"/>	Times
<input type="checkbox"/>	Severities
<input type="checkbox"/>	Phases
<input type="checkbox"/>	SizeDefs
<input type="checkbox"/>	Sizes
<input type="checkbox"/>	Checklists
<input type="checkbox"/>	Patterns

Mail Server:	celine.ics.hawaii.edu
To:	cmoore@hawaii.edu
Subject:	Review: Multi-User Calendar Design
Comment:	
Attached Data:	0117151744.drl

Send Cancel

Figure 3.23. Email Review Data window. The reviewer is sending their defect data to Cam. Only the defect data will be attached to the email message.

Leap toolkit attaches a leap data file to the email message when it is sent. The last thing the reviewer must do is select the data that will be sent in the email. Since it is a review, the reviewer just sends the defect data. When they press Send, the Leap toolkit builds an email message and sends it to Cam.

Once Cam receives all the reviews from Robert, Joe and Tie, he starts Io to record time for the Design phase. Cam's development process does not have a design review or code review phases. He considers review to be a part of the main phase. He loads the three defect files into the Leap toolkit and reviews all the defects that were found. If he agrees that the defect is really a defect he can fix it and record the amount of time it took to fix. If he doesn't think that the defect is a real defect he can delete the defect. When all the defects found in the design review are fixed, Cam stops Io, records the time and moves on to the coding phase.

**Coding** Cam uses Io to record the time he spends coding. While Cam is coding the program, he records each requirement and design defect he finds and the amount of time it takes him to fix it. For example, when he realized that the Permission class needed an additional private method to set the level he switches to Mano and records the defect (see figure 3.24). He clicks on the start time in Mano and returns to fixing the defect. When he has fixed the problem, he switches back to Mano and hits the stop button to record the amount of time it took to fix the defect. He then records the defect.

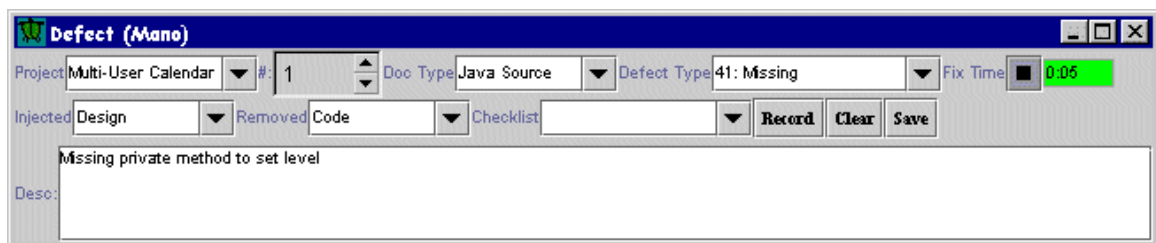


Figure 3.24. Defect Recording Tool, Mano, showing the design defect found during coding. Cam has just started to fix the defect. The timer has just started.

As Cam is coding the program he must be aware of interruptions and record them in Io. If his phone rings or a colleague comes into ask him a question he clicks the pause button in Io. When he gets back to work, he un-pauses Io.

**Code Review** When Cam finishes coding the entire project he stops Io and begins a personal Code Review. He decides to conduct a personal code review since he believes that his colleagues are too

busy to help him in a timely fashion and he believes that code reviews tend to find less interesting defects. He opens the Checklist table and filters for Code Review checklist items and starts the code review. Figure 3.25 shows Cam's Code Review checklist. He reviews all the code in the project.

	Checklist	DocType	Phase	Description
1	11: Misspelling	Java Source	Code	Check all identifiers for misspellings/miscapitalizations.
2	12: Static	Java Source	Code	Are any static methods called with an instance variable? Are any non-static instance variables referred to outside an instance?
3	13: Braces	Java Source	Code	Is the code "brace'd" correctly? Check each method and the end of the class.
4	14: Semi-colon	Java Source	Code	Is each statement terminated with a ";"?
5	15: Parentheses	Java Source	Code	Are all "("'s matched in all casts and conditionals?
6	21: Imports	Java Source	Code	Are all necessary packages imported?
7	41: Conditionals	Java Source	Code	Check each conditional test. What kinds of data values are possible in the test terms? Does the test operate correctly under all cases?
8	42: While Loops	Java Source	Code	Check each while loop. Is there a "counter"? Is it changed within the loop? Is it guaranteed to make the test return false eventually?
9	43: For Loops	Java Source	Code	Check each for loop. Does the index start and stop on the right values?

Save Load Clear Table Add Rows Delete Ro... Hide Rows Show All ... Close

Loaded U:\Leap\cam-check.leap.

Figure 3.25. Cam's Code Review Checklist.

Cam ignores uses checklist item 6, "21: Imports", because based upon his historical data it is easier to fix the defect when the compiler finds it than in code review. To figure out if he is missing an `import` statement during code review takes a lot of effort, but during compile the compiler tells him almost immediately. When he finds a defect that he thinks is important to record, he records the defect using Mano and fixes the defect. He records the fix time by using Mano's timer. When he finishes reviewing all the code he stops Io. After finishing the code review Cam moves on to the next phase of his development process, Compilation.

Cam believes that he is more productive if he completes all his coding before he compiles. When he learned the PSP he was forced to use this process, but based upon two small experiments that he conducted with his own development process, he now feels that he is more productive when he waits to compile.

**Compile** He starts Io to record the Compile phase. Cam fixes all the syntax errors that the compiler finds but does not record them. Since based upon his historical defect data, they are uninteresting and take very little time to fix. When he finds an interesting defect, he records it Mano. When he

finally gets a clean compile he is finished with the compile phase. He stops Io and starts the next phase of development, Testing.

**Test** He starts Io for the test phase and starts running all his tests. For each defect found during test Cam switches to Mano and records the defect. When the program passes all the tests, Cam stops Io and moves to his last phase of development, Postmortem.

**Postmortem** Cam starts Io to record time for the Postmortem. First, he runs LOCC on all the source files to determine the size of the Multi-User Calendar project. Next, he loads the Leap data file that LOCC produces into the Leap toolkit and checks to see that all the size data is associated with the correct project. If they are not he changes the list of files associated with the project to match the source files. When all the project data is loaded, Cam condenses the data using Hee. During condensation, the Leap toolkit summarizes all the raw data and calculates comparisons between the project plans and the actuals. Once the data is condensed, Cam compares his plans against what really happened. Figures 3.26, 3.26, 3.26 and 3.26 show the tabular comparisons between the estimates and the actuals. After looking at Hee, Cam takes a look at his time planning

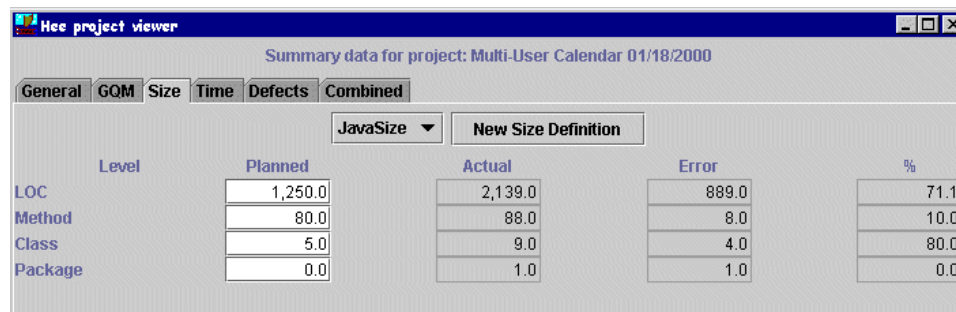


Figure 3.26. Hee project summary showing the Multi-User Project size estimates and actuals. Cam was close in his size estimation.

accuracy. The time planning accuracy chart shows him the ratio of planned time / actual time for all his projects. Figure 3.30 shows this graph. The most recent projects are on the left of the chart. According to this graph Cam's planning accuracy is oscillating between 150% over planning and 50% under planning. Cam's not too happy with this, but it looks like this last project is much better and he hopes to keep up the good work.

Cam also takes a look at his defects by charting the number of defects per defect type against the amount of time it took to fix those defects. Figure 3.31 shows the results for the Multi-User Calendar project. Cam notices that his failed test cases took the most amount of time to fix.

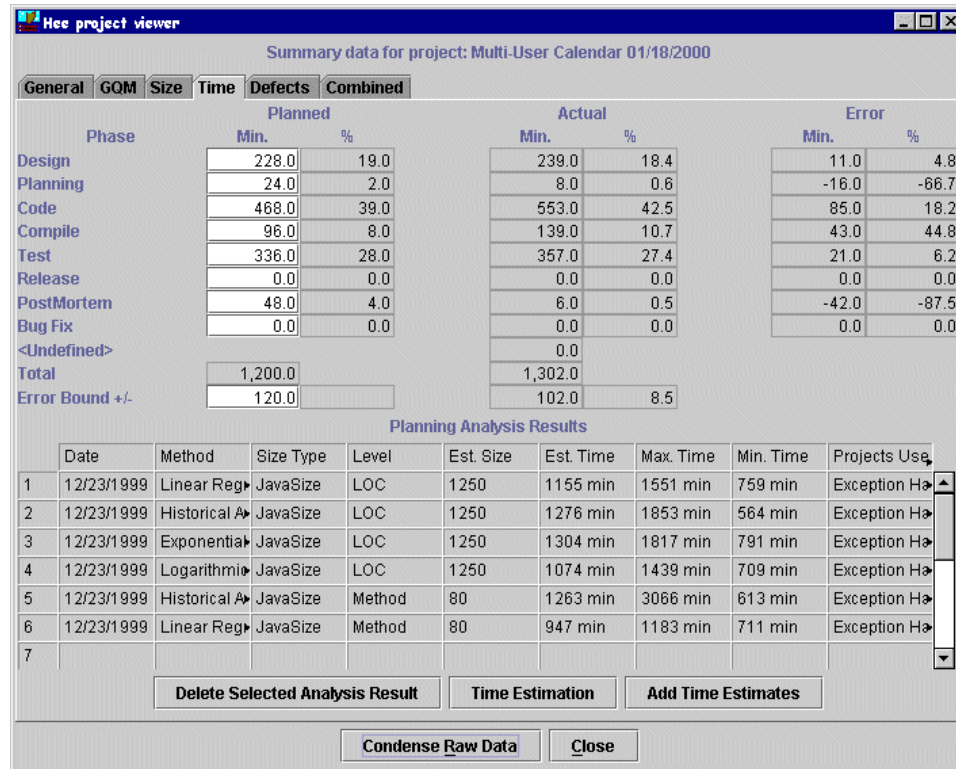


Figure 3.27. Hee project summary showing the Multi-User Project time estimates and actuals. Cam was very close in his time estimation. He underestimated the amount of time by less than 10%.

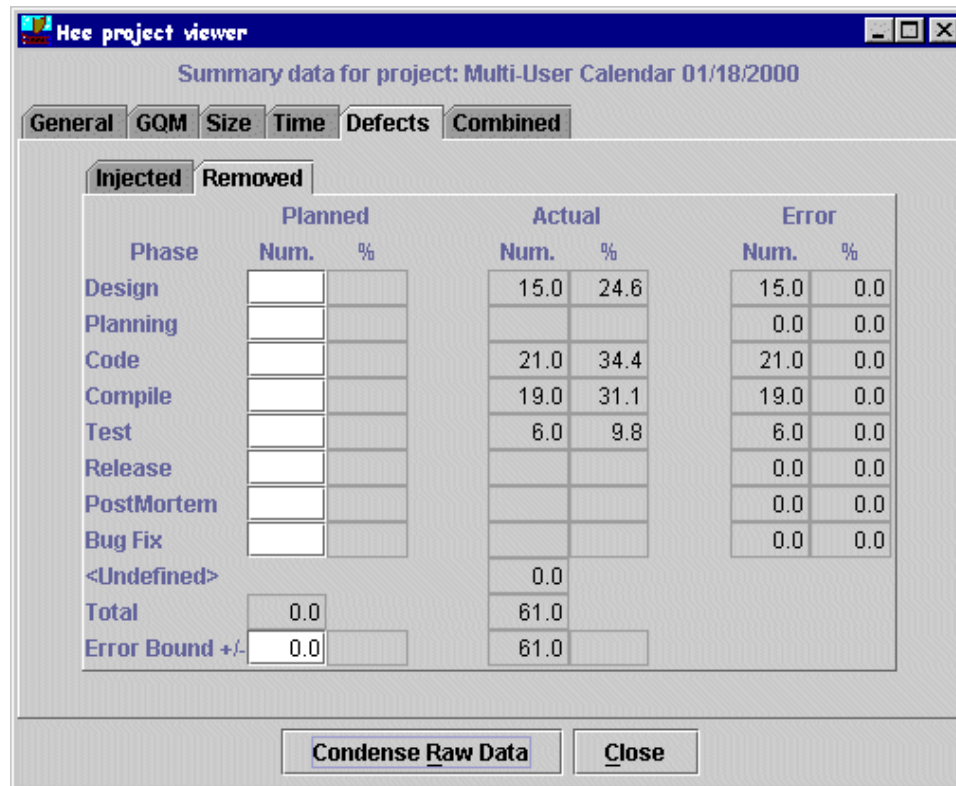


Figure 3.28. Hee project summary showing the Multi-User Project defects removed. Cam removed 59% of the recorded defects before the compile phase.

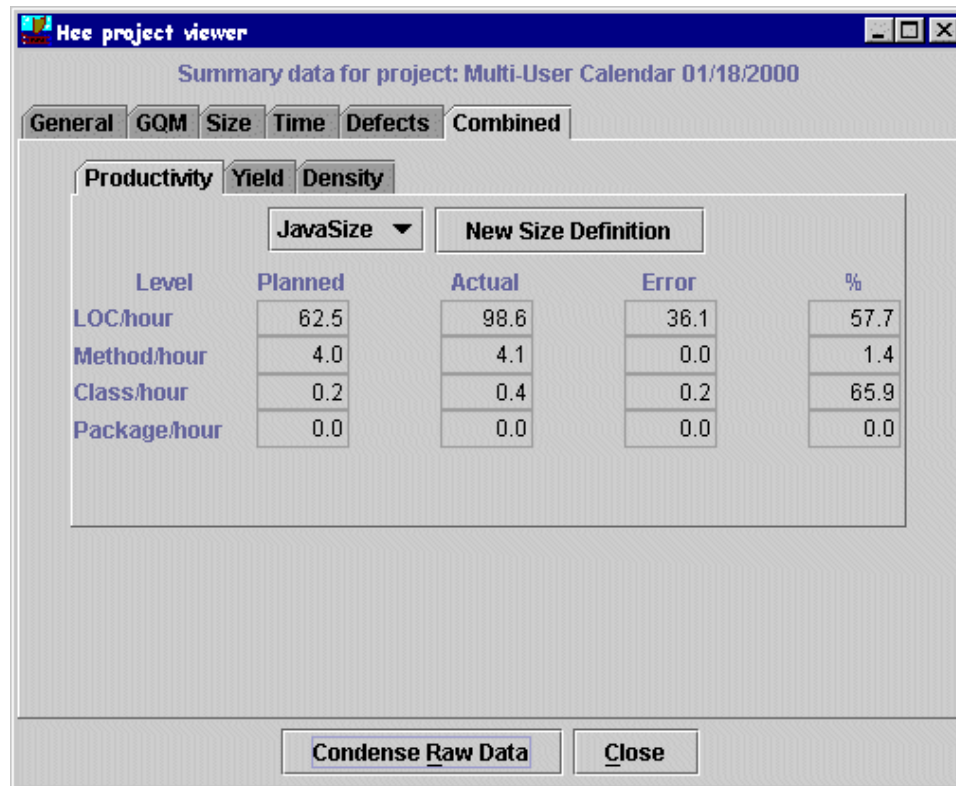


Figure 3.29. Hee project summary showing the Cam's productivity. Cam worked much faster than he expected.

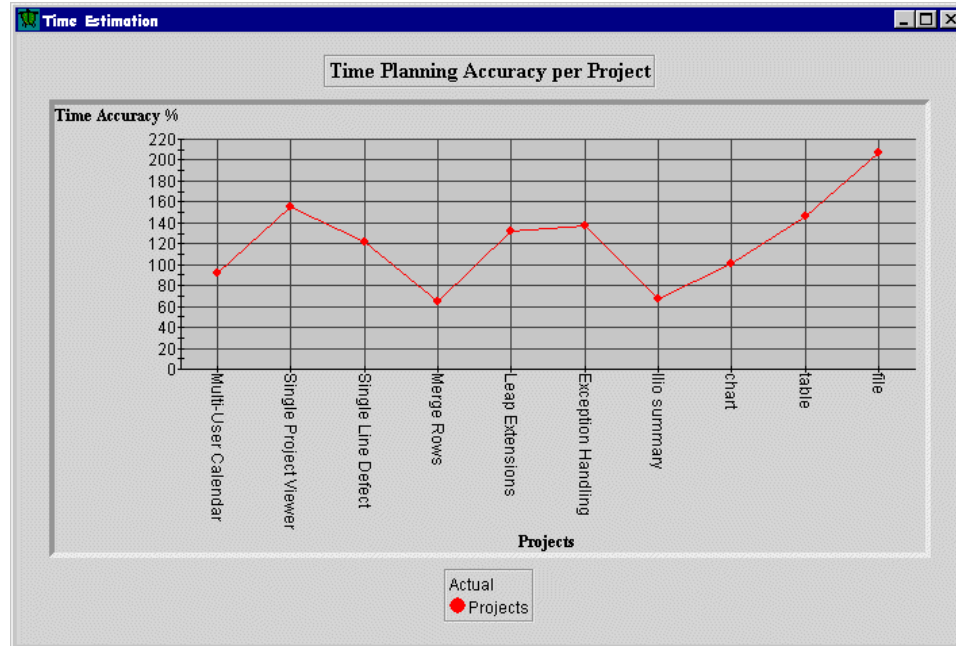


Figure 3.30. Time Planning Accuracy chart. Cam's planning accuracy has been oscillating between 50% and 150%.

This is pretty obvious since Cam had to track down why the test case failed and then had to fix the error without introducing new defects. Two other defect types stand out, 40: Method and 64: Null Pointer Exceptions. For the 40: Method defect it also makes sense that if Cam's design were missing a method or added methods that were not necessary then fixing the defect would take more time since more code would be affected by these types of errors. The 64: Null Pointer Exception defects are very interesting to Cam. He reflects on what caused this type of error and remembers that he didn't check the return values of method calls before he added the object to other data structures. If the method returned null then the other data structures were corrupted and the Null Pointer Exception would be thrown when the program ran. Cam decides to pay more attention to the return values of his methods and check for null more often. Hopefully, this will improve the quality of future projects.

### 3.6 Comparing the Leap toolkit example and the PSP example

In this section, I will compare the PSP example from Section 2.1.3.1 and the Leap toolkit example. Both examples used the same project and developer, the only difference was in the first



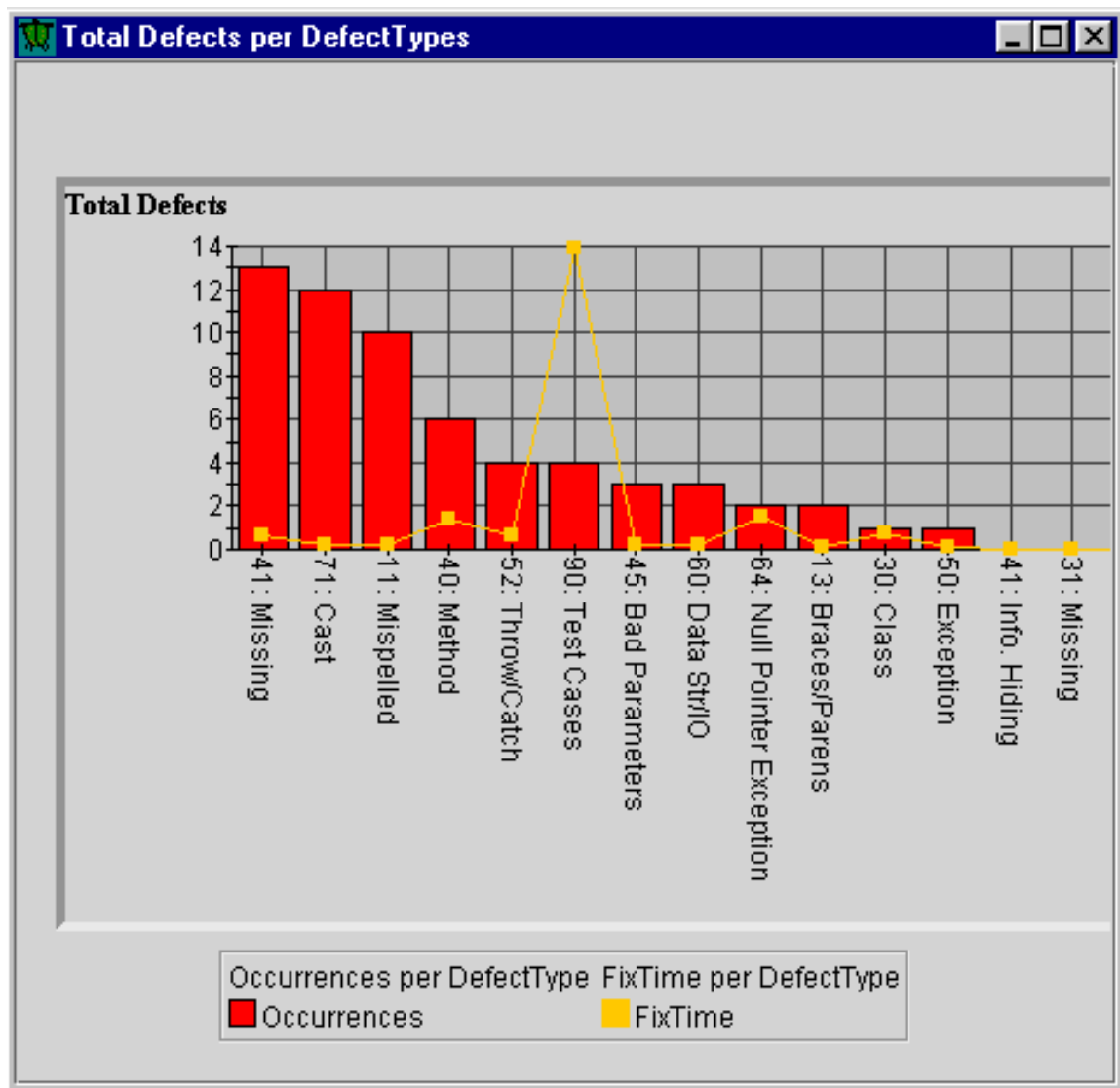


Figure 3.31. Defect analysis, number of defects and the total amount of time it took to fix them.

example Cam used PSP 2.1 to guide the development and in the second example he used the Leap toolkit.

### 3.6.1 Planning new projects

Accurate project estimation is an important goal of both the PSP and the Leap toolkit. The PSP and the Leap toolkit both support historically based project estimation. In the PSP, the developer manually calculates the correlation and the linear regression coefficients. To calculate the linear regression coefficient, the developer uses the following formula:

$$r(x, y) = \frac{n \sum_{i=1}^n x_i y_i - \sum_{i=1}^n x_i \sum_{i=1}^n y_i}{\sqrt{\left[ n \sum_{i=1}^n x_i^2 - \left( \sum_{i=1}^n x_i \right)^2 \right] \left[ n \sum_{i=1}^n y_i^2 - \left( \sum_{i=1}^n y_i \right)^2 \right]}}$$

Where  $x_i$  is the size data for each project and  $y_i$  is the actual time for each project. When a developer uses the PSP they must calculate the  $r^2$  value by hand. They have to gather all their previous project summary forms and calculate the correlation between planned sizes and actual times. If it is too low then they check the  $r^2$  value for their actual sizes to actual times, again using the above formula. If their size and time data are correlated then they use a linear regression to calculate the time estimate. The formula is

$$\text{Time estimate} = \beta_0 + \beta_1 \text{size estimate}$$

The coefficients  $\beta_1$  and  $\beta_0$  are calculated using the following equations:

$$\beta_1 = \frac{\sum_{i=1}^n x_i y_i - n x_{avg} y_{avg}}{\sum_{i=1}^n x_i^2 - n x_{avg}^2}$$

$$\beta_0 = y_{avg} - \beta_1 x_{avg}$$

The developer calculates these coefficients by hand. This process may take a few minutes. Since the calculations are not trivial, there is a good chance that the developer will make a mistake and get bad results.

A developer using the Leap toolkit does not manually calculate any of the values. The Leap toolkit's Time Estimation Tool calculates all the values and provides a graphical representation of the size and time data. Figure 3.32 shows the Time Estimation Tool. The  $r^2$  value for Cam's planned size in methods to actual time is 0.8623 and the  $\beta_0$  is 94.93 and  $\beta_1$  is 10.66. The time estimation tool has calculated the estimated time for a size estimate of 80 methods. Cam had to choose the historical data set, trend line, the grainsize and the size estimate, the Time Estimation tool did all the calculations. The Hee project tool allows Cam to compare different estimates from

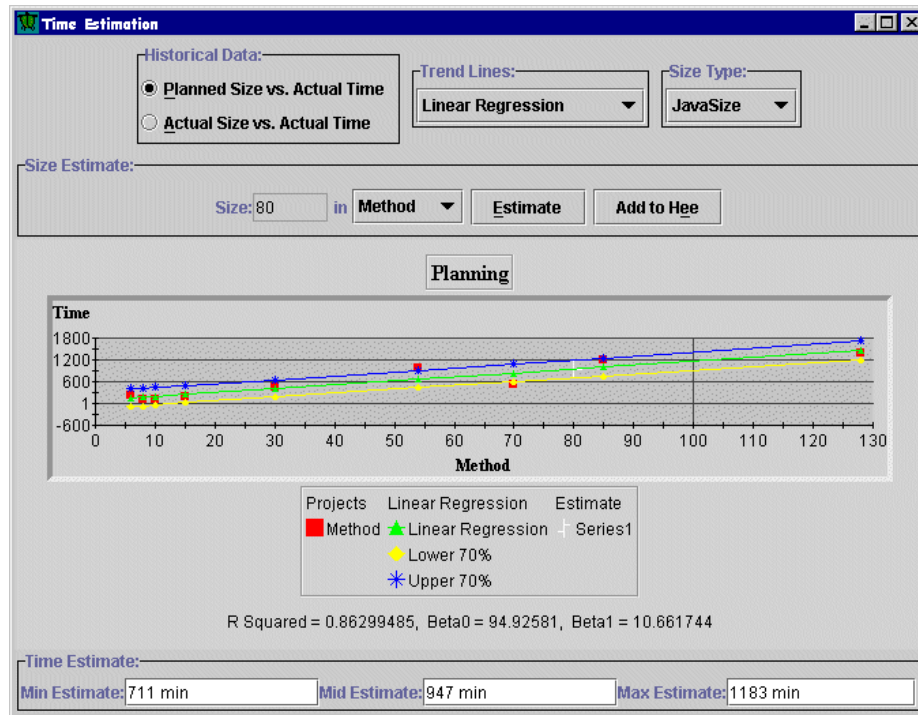


Figure 3.32. Time Estimation Tool performs all the calculations for Cam.

estimation methods. He can see what different models predict and choose the estimate that he feels most comfortable with. This whole process takes a matter of seconds.

### 3.6.2 Recording process data

There is a great difference between collecting data using the PSP and the Leap toolkit. Since the PSP is a manual process, every time the developer needs to record data they must stop their work and shift their focus to a paper form. This greatly interrupts their software development. Once they have found the right form and something to write with, they can enter the data. After the data is collected they shift back to working on their computer.

Since the Leap toolkit is a program running while the developer is working, it greatly reduces the overhead of recording process data. When the developer needs to record data, they can easily switch the window focus and enter the data. The Leap toolkit also provides default values for many of the fields that the user must fill in. This also reduces the overhead of data recording.

Another advantage of the Leap toolkit over the PSP is the Leap toolkit automates much of the calculations. In the PSP example, I used the following time entry shown in figure 3.33. Cam

**Table C16a Time Recording Log**

Student		Cam Moore				Date	3/13/00
Instructor						Program #	23.5
Date	Start	Stop	Inter. Time	Delta Time	Phase	Comments	
3/13	9:34	10:02		28	Planning	Multi-user calendar extension	
3/14	9:12	10:55	26	67	Design		

Figure 3.33. Time Recording Log after Cam goes to the staff meeting.

made a simple math error when he calculated the delta time for the second time entry.  $10:55 - 09:12 = 103$  minutes.  $103 - 26 = 77$  minutes. The correct delta time should have been 77 minutes not 67. This type of data was observed frequently in research on PSP data quality by Disney and Johnson[4] The Leap toolkit does all the math for the developer and would not make this mistake.

### 3.6.3 Flexible process definition

In the PSP example Cam followed the PSP2.1 process for development, where he does planning before design. The first step in the planning process is to develop a conceptual design for the project. The conceptual design is used for developing the plan. Once the plan is complete the developer goes back and finishes designing the project. Major changes in the design may have large affects on the plan, but the planning phase is complete. In the Leap toolkit example Cam made design the first step in his process to help him develop a better plan. The Leap toolkit allows the user to define their own development process and follow it. This allows developers to use the Leap toolkit for other non-programming activities like report or paper writing.

### 3.6.4 Conducting reviews

The PSP is solely focussed on the individual developer. No where in the PSP does the developer get feedback from other developers. The individual developer does all the data collection and analysis. The developer using the PSP conducts modified reviews. Instead of having other developers review their designs and code, the developer alone conducts the review. Conducting an individual review greatly reduces the management overhead of the review, but also reduces the range of feedback. Without other reviewers' points of view the review breadth is limited.

I believe the most insightful feedback comes from other developers reviewing your work. I designed the Leap toolkit to support group review, to help the developer get this feedback. In the Leap example, Cam asked his fellow developers to review his design for the project. The Leap toolkit's ability to load data files from URLs and email data files greatly reduces the management overhead of conducting a review. Since the other developers used the Leap toolkit, Cam quickly gathered and analyzed the reviewers' defects. The Leap toolkit also allows Cam to build a database of all the defects found in his software — both by him and by reviewers.

### **3.6.5 Conducting the postmortem**

In the two examples the postmortem phases look very similar and complex. In the PSP the postmortem phase is complex and time consuming. The developer must go back through all their data logs, collate the data and enter the values on the Project Plan Summary form. This may take some time to add up all the time spent per phase or the number of defects injected in Code. Calculating the historical “To Date” values can also be time consuming. All of this work must be done by hand.

The Leap toolkit automates most of this work. The developer ensures that they have all their data loaded. Then they press a button to condense the raw data. Condensing the data creates a summary of the project that is stored separately from the raw data. The project summary combine all the size, time and defect summaries. The Leap toolkit uses these summaries for the historical data used in planning. The project summaries reduce the total amount of data that must be loaded to do planning. The condensation process takes a few seconds. Once the data has been condensed the developer can view many different analyses without performing any more computations. The developer using the Leap toolkit can focus on the results of the analyses not performing the analyses.

## **3.7 Benefits of Leap toolkit's design**

I designed the Leap toolkit to be flexible and easy to use, while supporting developer improvement. Three important benefits of the Leap toolkit are: (1) it addresses data errors, (2) it addresses collection errors by reducing the overhead of data collection, and (3) it provides data analyses that are not practical with a manual method.

### 3.7.1 The Leap toolkit addresses many important classes of data errors found in the PSP

The Leap toolkit tries to address each category of data error found in Anne Disney's study. Disney's research classified the data errors into seven categories. The architecture, design and implementation of the Leap toolkit addresses each of these categories.

- **Calculation Error:** This error type applies to data fields whose values are derived using any sort of calculation where the calculation is done incorrectly. In her study, 46% of the errors that Disney found were calculation errors. To eliminate this type of error, the Leap toolkit automates several different types of calculations. Some of them are durations, regressions, and summarizations.

The Leap toolkit has a class called `TimeInterval` that represents durations. The `TimeInterval` class provides methods to create new `TimeIntervals` given a start and stop date or number of milliseconds. It also has a method to subtract two `TimeIntervals`. The Leap toolkit, specifically Naia, uses the `TimeInterval` class to calculate the duration of a time entry from the start, stop and interrupt times. In the PSP example, I used the following time entry shown in figure 3.34. This time entry has a simple calculation error in it. Cam failed to

**Table C16a Time Recording Log**

Student		Cam Moore				Date	3/13/00
Instructor						Program #	23.5
Date	Start	Stop	Inter. Time	Delta Time	Phase	Comments	
3/13	9:34	10:02		28	Planning	Multi-user calendar extension	
3/14	9:12	10:55	26	67	Design		

Figure 3.34. Time Recording Log after Cam goes to the staff meeting.

properly calculate the duration for the Design phase.

$$10 : 55 - 9 : 12 = 103\text{minutes}$$

$$103 - 27 = 77\text{minutes}$$

Figure 3.35 shows the same time entry with the correct duration calculated by the Leap toolkit.

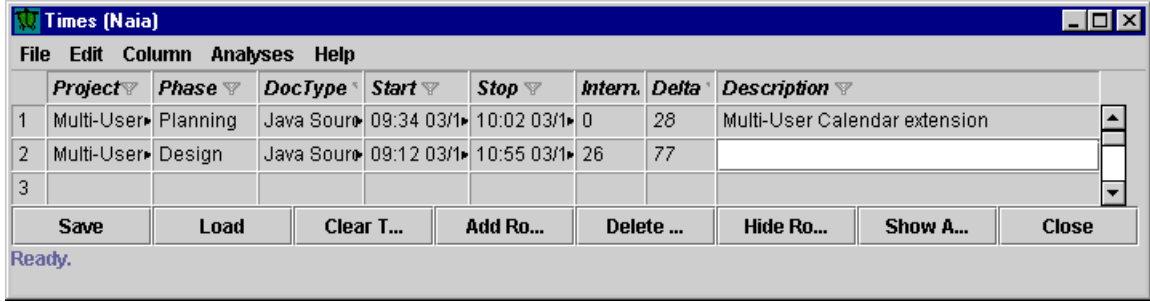


Figure 3.35. Naia, the time recording tool, with the correct duration .

Another type of calculations automated by the Leap toolkit is regression. The PSP recommends the developer calculate the correlation coefficient  $r^2$  for their size and time data to determine which set of data to use in estimation. The developer must solve this equation

$$r(x, y) = \frac{n \sum_{i=1}^n x_i y_i - \sum_{i=1}^n x_i \sum_{i=1}^n y_i}{\sqrt{\left[ n \sum_{i=1}^n x_i^2 - \left( \sum_{i=1}^n x_i \right)^2 \right] \left[ n \sum_{i=1}^n y_i^2 - \left( \sum_{i=1}^n y_i \right)^2 \right]}}$$

and then square the result for many different sets of data. This is a difficult equation to solve by hand. The Leap toolkit has several classes that help calculate different regressions: LinearRegressionData, ExponentialRegressionData, PowerRegressionData, LogarithmicRegressionData, and DataSeq. Given two sets of data, these classes automatically calculate the  $r^2$  value for the data. The Leap toolkit uses these classes in the Time Estimation Tool (see Fig 3.16).

The Leap toolkit also automates the time consuming process of summarization. It provides many classes and features that summarize the raw size, time, and defect data for the user. The Moa tool summarizes all the projects that the user chooses. It totals the size, the time spent per phase, and the defects made per phase for the projects. By calculating the totals for the user, the Leap toolkit eliminates addition and categorization errors made by users in the Disney study. For example, a Leap toolkit user sent me a bug report where they said the Leap toolkit was not summarizing the number of defects correctly. They said they had 39 defect entries and the Leap toolkit was telling them they made 54 defects. When I looked at the user's Leap data, I found that the user was only counting the number of defect entries while the Leap toolkit was correctly adding up the number of *occurrences* of each defect. These

two numbers are not necessarily the same, since a single defect entry could represent 2 or more occurrences of a defect. The Leap toolkit's total was the correct answer.

- **Entry Error:** This error type applies to fields where the user clearly does not understand the purpose of the field or used an incorrect method in selecting data. Disney found that 9% of the errors were entry errors. The Leap toolkit provides two mechanisms to reduce entry errors: default values and choices from defined values.

When the user selects a cell in a new row, the Leap toolkit fills in some of the values with default values. This simplifies data entry for the user. There are three different types of default values in the Leap toolkit: *fixed*, *last visited* and *date*. *Fixed* default values are values that do not change. For example, the occurrences column in Honu, the defect table, defaults to 1. For most defects this value is correct, so the user does not have to enter the value. The other column that has a fixed default is the FixTime column in the defect table. It defaults to 1 minute. The *last visited* default value inserts the last value that was used in the column. For example, once a user selects “Multi-user Calendar” as the project, the Leap toolkit will use that value to automatically fill the project field on subsequent entries until the user selects a different value. Other columns that use the last visited default are Phase, DocType, and DocID. The *date* default value inserts the current date and time into the cell. This default is used for the start and stop times for Naia (the time table) and Ilio (the project table). Figure 3.36 shows an example of the default values in Naia. The Leap

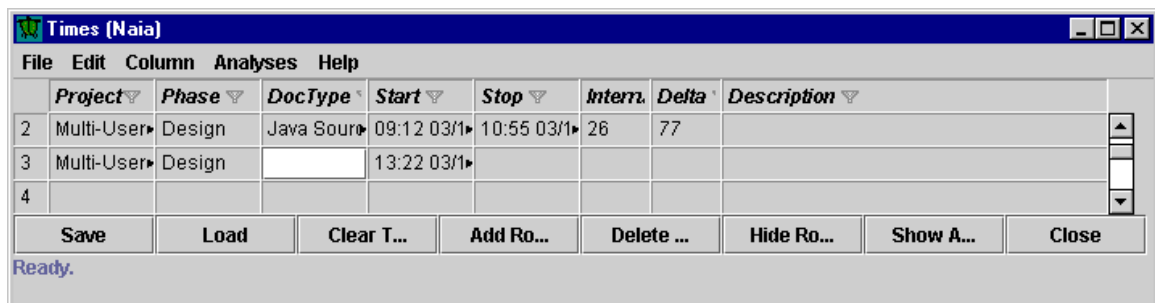


Figure 3.36. Default values inserted into Naia: Project, Phase, and Start date.

toolkit inserted the project, phase, and start time when the user clicked into the DocType column. As a result, the only field the user has to explicitly choose is the document type for this time entry. When they finish working they can select the Stop field and the Leap toolkit will automatically insert the current date and time.



In addition to the default values, the Leap toolkit provides pop-up menus with defined values for many of the important fields. This allows the user to choose from defined values, reducing the chance that they will incorrectly fill in a field. Figure 3.37 shows an example of the pop-up menu for the defined projects in the Time editing table Naia.

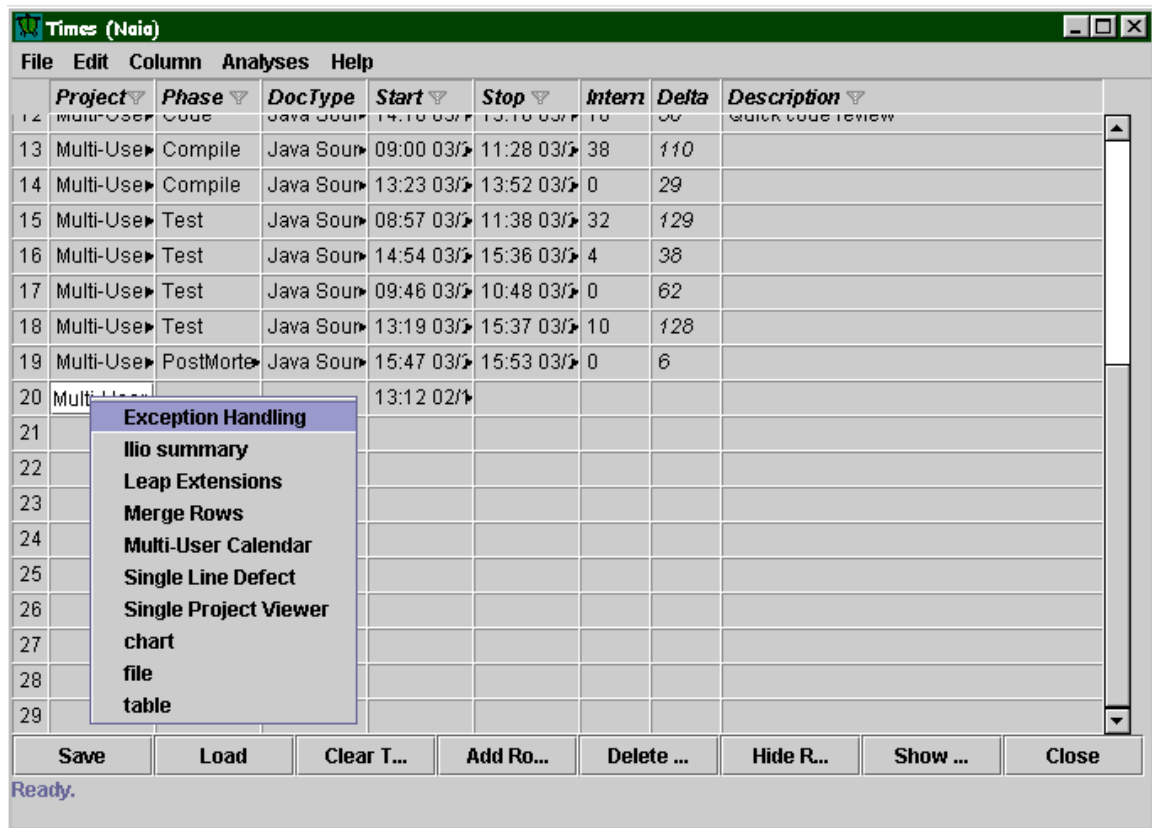


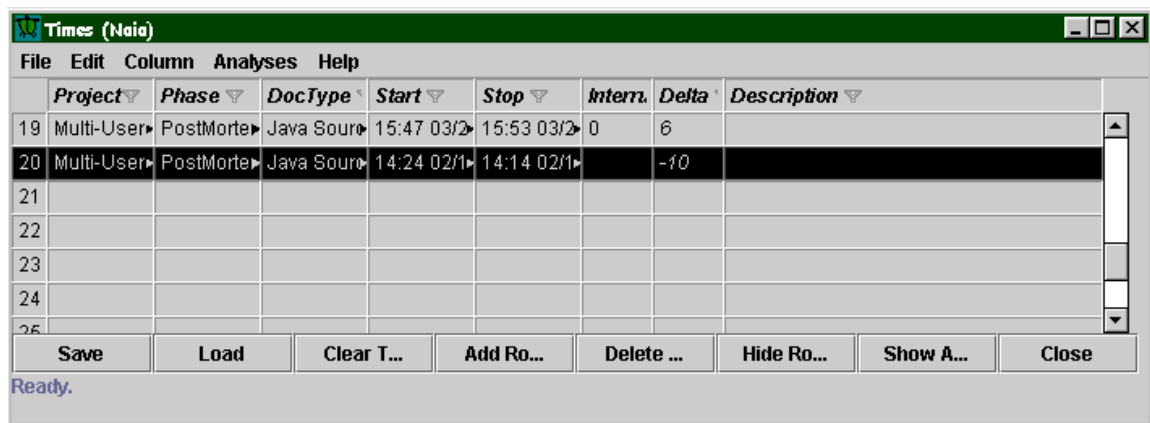
Figure 3.37. Pop-up menu for the currently defined projects.

The Leap toolkit's calculation features also reduce the opportunity for entry error. Since the user does not enter values into calculated fields, they cannot make entry errors.

- **Sequence Error:** This error type is used to indicate when the user incorrectly moved back and forth between phases in violation of the specified development process. Only 1% of the errors found in Disney's research were sequence errors. One of the design principles behind the Leap toolkit is **Light-weight**. The Leap toolkit should not impose heavy constraints on the developer. Therefore, the Leap toolkit does not enforce a specific development process on the user. If the user wants to switch between phases of development, the Leap toolkit will total all the values for the phases and consider them one phase. For example, if a user works

on design, then develops a project plan and estimate and then returns to complete the design work, the Leap toolkit will view the two design entries as part of a single phase. If the user decides that they want more detail in the analysis, then they can define an initial design phase and a detailed design phase. Users can choose their own development strategy and build their own development practice.

- **Impossible Values:** This error type indicates that two values were mutually exclusive. Disney found that 6% of the errors were impossible values. To detect impossible values, the Leap toolkit has a rudimentary consistency checker for some types of data. This checker indicates to the user when data values are “impossible”. In the time table, Naia, time entries with a stop time that is before the start time are displayed in red. Figure 3.38 shows a time entry with a stop time that is before the start time. Since the image is black and white, I highlighted row



	Project ▾	Phase ▾	DocType ▾	Start ▾	Stop ▾	Intern	Delta ▾	Description ▾
19	Multi-User	PostMorte	Java Sourc	15:47 03/2	15:53 03/2	0	6	
20	Multi-User	PostMorte	Java Sourc	14:24 02/1	14:14 02/1		-10	
21								
22								
23								
24								
25								

Save Load Clear T... Add Ro... Delete ... Hide Ro... Show A... Close

Ready.

Figure 3.38. Time table (Naia) with a time entry that is in error.

20. If the image was in color you would see that row 20 is red and the rest of the rows are black. This tells the user that there is a problem with the time entry. They can then fix the error.

Currently Leap does not look at overlapping time entries, but I could build a consistency tool to check for this situation. The two timers Io and the fix time timer in the single line defect entry tool help eliminate this defect. By using Io consistently the user cannot produce overlapping time entries. They can create a time entry with negative time, but it will still be shown in red in Naia so they can easily see that it is in error. The timer in the defect entry tool will reduce the chance of incorrectly recording the amount of fix time for the defect.

The Leap toolkit does not totally eliminate all forms of impossible values. But it does greatly reduce the chance that they occur and in some cases indicate where there are problems.

- **Blank Fields:** This error type applies to data fields that are required but not filled in. Disney found that 18% of all the errors in the study were blank fields. The Leap toolkit does not consider blank fields to be an error, but does provide support for identifying and correcting blank fields. If a blank field is needed for analysis then it is considered *<Undefined>*. If there is a blank number field (such as the number of defects) then the Leap toolkit uses the default value 0. When the user sees the *<Undefined>* value in a report or chart they can go back to the data to see why Leap thinks the data is undefined. This helps the user gain better insight into their data. For example if there are defects recorded with a defect type that is not defined in the defect type table, then during analysis those defects are given the type *<Undefined>*. Figure 3.39 shows an example analysis of defect by defect type. In this example, the Leap toolkit found 22 defects that did not have a defined defect type. The user can either look at the defects and enter a defined defect type for them, or they can define a new defect type for the defects.

- **Intra-Project Transfer Error:** This error type applies to data fields whose values involve other data fields in the same project, but are incorrectly filled in. Disney found 6% of the errors were intra-project transfer errors. The Leap toolkit addresses this issue by using the raw size, time, and defect data for most of the data analyses. The Leap toolkit's Model-View-Controller architecture provide access to the `LeapDataSources` that hold the raw data. Thus, there is no possibility of a "transfer" error, since all tools operate on the same data repository.

For project based analyses, the Leap toolkit uses a condensation process. The condensation process condenses all the raw data for a given project into a summary of the project. The Leap toolkit uses the raw time, size and defect data to calculate these condensations. This summary is stored in the project table. The user decides when to condense the project. If new raw data is added for the project, then the user can re-condense the project. Since the Leap toolkit handles all the data transfers, there cannot be transfer error due to human error.

- **Inter-Project Transfer Error:** This error type applies to data fields whose values require data from a prior project but where the value used is not the same as the prior project's value. Inter-project transfer errors accounted for 14% of the errors in Disney's study. The Leap toolkit eliminates this type of error by handling all the data transfer issues automatically. When a

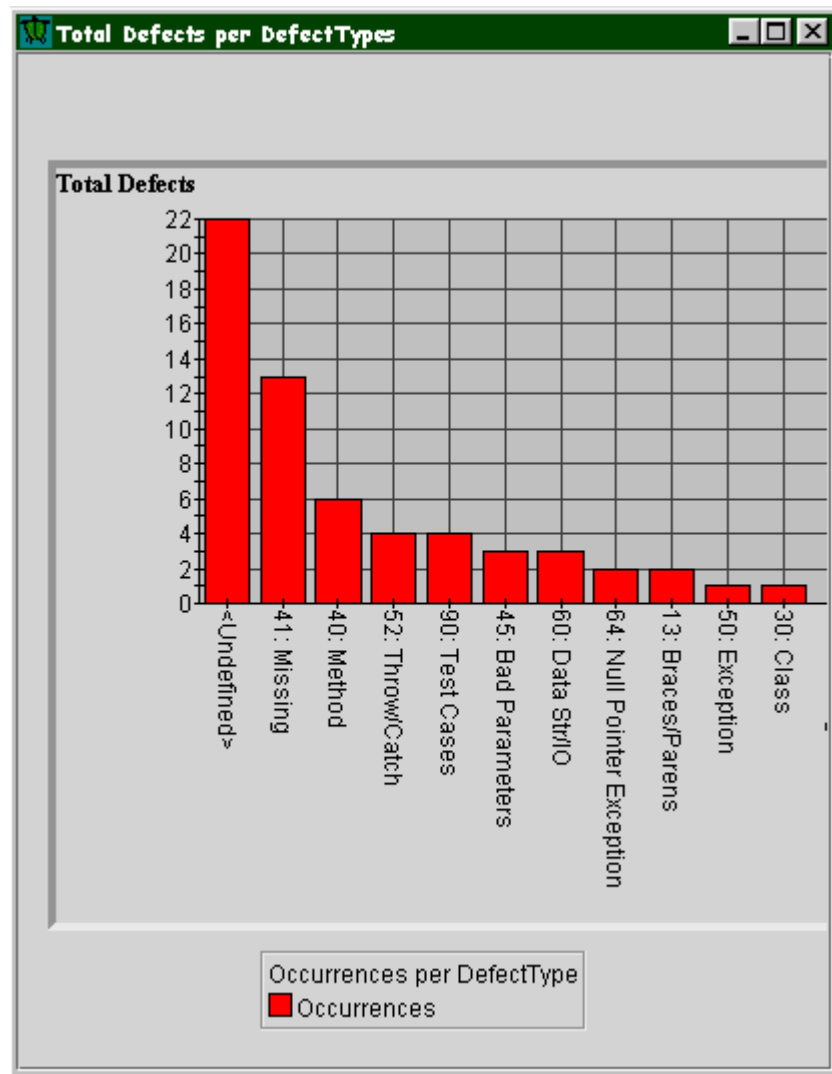


Figure 3.39. The Leap toolkit groups all undefined values into the *<Undefined>* category.

value is needed from another project, the Leap toolkit looks at that project's data condensation to find the value. The totals and percentages that are normally transferred from one project to the next are not transferred in the Leap toolkit. Instead, the totals and percentages are calculated from the loaded data. For example, consider the calculation of the percentage of the time spent per phase. In a manual system, the user would take the totals and percentage values from the most recent project and add the values for the current project to get the totals for all the projects. If they copy incorrect values from the previous project, then their future results will be in error. This cannot occur in the Leap toolkit since it automatically calculates all the totals and percentages. In addition, the Moa project summary tool allows the user to choose which projects they wish to summarize. Figure 3.40 shows the project selection dialog box. The totals are calculated from the condensed data for the selected projects and

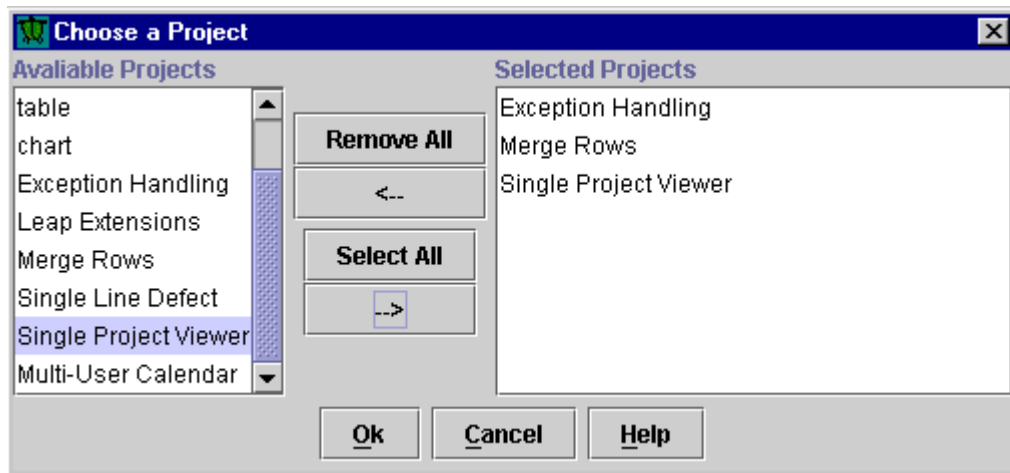


Figure 3.40. Moa, Project Comparison tool project selection screen.

presented.

The Leap toolkit addresses the data quality issues raised in Anne Disney's research of the manual PSP. Many of the categories of data error are eliminated and cannot occur in the Leap toolkit. The other categories of data error, entry error and impossible values are reduced by providing default values and rudimentary consistency checking. By addressing and eliminating many of the types of data error, the Leap toolkit provides improved data quality over manual software developer improvement methods.

### **3.7.2 The Leap toolkit addresses collection stage errors by reducing the overhead of data collection**

Automated support for entry removes simple data entry error. For example the user does not have to write down the time that they start working. This reduces the chance that they make a mistake. Also the Leap toolkit displays the current elapsed time. This feedback allows the user to check and see if the Leap toolkit is accurately recording what is happening.

Since the Leap toolkit lowers the user's overhead, it should reduce collection stage errors. The user is more likely to collect accurate data if it is easy to collect. High overhead will cause the user to not bother collecting data. Also the ease of analysis shows the user the benefit of accurate collection of data. This should motivate them to collect good data.

### **3.7.3 The Leap toolkit provides data analyses that are not practical with a manual method.**

The Leap toolkit is designed to support many different methods of estimation and planning. For estimating size, the Leap toolkit supports multiple size representations. The user may choose a size representation that best fits their development process. They can experiment with their size estimation abilities by using different sizes and seeing which is best for them. For example a developer can estimate the number of function points and methods that a project will be. When they complete the project they can see which estimate was more accurate.

For time estimation based upon size estimate, the Leap toolkit supports multiple estimation models. The user can choose between averages, linear regression, exponential regression, power regression and logarithmic regression. The Leap toolkit allows the user to use their planned size values or their actual size values when making an time estimate. This flexibility allows the developer to find their best method of time estimation.

The Leap toolkit also allows the user to filter their data. This allows the user to match their historical data to the current project. By matching similar projects the user's estimates should be more accurate.

The next chapter discusses how I plan to investigate the last two benefits of the Leap toolkit.

# Chapter 4

## Investigating Individual Software Development

*There's a large journey to be taken, of many trials.* - Joseph Campbell

The main thesis of this work is that the Leap toolkit provides a novel tool that allows developers and researchers to collect and analyze software engineering data.

I used the Leap toolkit to investigate two important issues in software development, collection error and project estimation.

- The Leap toolkit addresses the problem of collection stage errors by reducing the overhead associated with collection.
- Since the Leap toolkit provides data analyses that are not practical with a manual method, I could compare the accuracy of different time estimation methods.

The remainder of this chapter is organized as follows. Section 4.1 details the case study I used to investigate collection errors. Section 4.2 describes the empirical investigation into planning accuracy.

### 4.1 Investigating collection stage errors

To investigate collection stage errors, I conducted a case study[43] of 16 graduate students in ICS 613. Most of the graduate students were in their second year of the Information and Computer Sciences Masters program. On average, they had taken over four graduate computer science classes.

They had a wide range of professional programming experience. One student reported having 20 years of programming experience and over eight years of professional programming experience. Another student reported only having 2 years of programming experience and no professional programming experience. The average for the class was 5.8 years of programming experience and 2.2 years of professional programming experience. Only three students reported having no professional programming experience.

Observing the collection stage errors is very difficult since direct observation of all the students was impossible. One purpose of this case study was to determine the level of data collection errors and the awareness of these errors by the students in the experiment. The results of the case study are presented in Section 5.2

#### **4.1.1 Case Study Method**

To find indications of collection errors I conducted four surveys, analyzed the students' Leap data, and interviewed 11 out of the 16 students in the class. I was unable to interview all the students because some of the students left Hawaii before I could schedule an interview. The answers to the surveys, the analysis of the students' data, and interview results allowed me to discover suspicious trends in the students' data. By combining the interview and survey data, I was able to gain insight into how the students used and felt about the Leap toolkit.

##### **4.1.1.1 Ensuring Anonymity**

To ensure anonymity of the surveys, I had each student place their survey in an envelope. They then sealed and signed the envelope. I did not open the surveys until after the grades for the class were turned in. This provided assurance to the students that the results of the surveys did not affect the students' grade in the class. Once I had all the surveys I assigned each student a random number and used that number as the student's identification.

#### **4.1.2 Data Collection**

There were three main sources of data in this case study: the student's raw data, the four surveys and interviews with the students.



#### **4.1.2.1 Student's raw data**

Each student in ICS613 turned in their Leap data for each project to Dr. Johnson. I got consent from all the students to access their Leap data. After the class ended I copied the Leap data files for analysis. I used the Leap toolkit to help analyze the data.

Some patterns in the Leap data may indicate collection error. Some suspicious trends in the data are having time data in increments of five or ten minutes or having times that start or end on the hour. The Leap toolkit records times in increments of seconds so it is extremely unlikely that anyone could work in exact increments of five or ten minutes. Such patterns indicate that the user is editing their data or recording their data after the fact and rounding. This is just an indirect indication that the data does not accurately reflect their actual development.

#### **4.1.2.2 Surveys**

Four times during the semester I had the students fill out a survey. I gave the first survey after the students had turned in the second programming assignment. I gave the second survey after the students had completed their first project where they had to estimate the size and effort for the project. I issued the third survey after the sixth programming assignment. I gave the final survey on the last day of class. Each survey asked some different questions. The next sections describe the focus of each survey. See Appendix A for the actual surveys that I gave to the students.

#### **4.1.2.3 Survey #1**

This survey was a baseline survey that gathered information about the students and their initial use of the time recording tools of the Leap toolkit. It focused on the student's programming experience, their experiences with time collection and any issues they had with the Leap toolkit.

The first section of the survey asked about the student's prior programming experience. The second section asked about their knowledge of software engineering principles. Dr. Johnson briefly covered many of these principles during the class. The third section asked the students about their time data collection experience. I asked which data collection tool they use more often. The fourth section asked the students to list three negative aspects about the toolkit and three positive aspects of the toolkit.

#### **4.1.2.4 Survey #2**

The second survey focused on usability issues of the Leap toolkit, the student's use of the time recording tools, and time estimation. At this point in the course the students had used the Leap toolkit's time estimation tool on one project. The usability portion of the survey looked at the human computer interface issues in the Leap toolkit. The second time survey allowed me to see if the students' perception or use of the time recording tools changed over time.

The first section of the survey asked about the Leap toolkit's usability. I was interested in improving the interface and reducing the overhead for the user. The second section asked the students about their time data collection experience. I found out which data collection tool they used most often. Combining the results of the first two surveys allowed me to see trends in tool usage. The third section asked them about their time estimation experience. I wanted to determine if making a plan affected their perception of the project. The fourth section asked the students to list three negative aspects about the toolkit and three positive aspects of the toolkit.

#### **4.1.2.5 Survey #3**

Survey #3 repeated the time collection, and time estimation sections from surveys #1 and #2. It also asked about the students' defect collection experiences. This survey focused on data collection and analysis. It looked at time and defect data collection and time estimation. The first three surveys allowed me to track trends in the students' tool usage and feelings about making estimates.

The first section of the survey asked the students about their time data collection experiences. The second section asked them about their time estimation experiences. The third section asked the students about their defect collection experiences.

#### **4.1.2.6 Leap Survey #4**

In the last survey I again asked the students to fill out a usability survey. I wanted to learn how their perceptions of the Leap toolkit changed over the semester. This survey focused on the students' perception of the Leap toolkit.

The first section of the survey asked about the Leap toolkit's usability. The second section asked the students about their perceptions of the Leap toolkit. The third section asked them about any lessons they learned from using the Leap toolkit.

### **4.1.3 Interviews with students**

In addition to the surveys and raw Leap data collection, I conducted interviews with eleven of the sixteen students. During these interview I asked the students about their feelings about using the Leap toolkit to collect data about their software development process. I wanted to determine how difficult collecting all the data was and if collecting the data changed the way they programmed. After these questions I asked about the accuracy of their Leap data, issues about making estimates and the overhead of using the Leap toolkit for improvement. The next portion of the interview focussed on measurement dysfunction issues. I asked if the students felt pressure to make their actual data match their estimates and if they compared their Leap data with the Leap data of other students. Finally, I asked them about what they learned about their own development process. See Appendix C for the raw transcripts of the interview questions and answers.

### **4.1.4 Data analysis**

I combined the results of analyzing the raw Leap data files, the surveys and the interviews to detect collection errors, evaluate estimation accuracy and detect other interesting issues with empirically based individual software developer improvement. Chapter 5 presents the results of the data analyses.

## **4.2 Investigating planning and estimation**

I conducted an empirical investigation to evaluate 13 different “mechanical” estimation methods and the students’ estimation method to determine if there is any significant difference between the estimation methods. All thirteen estimation methods make time estimates based upon historical size measures. The differences between the methods are the type of size measure used and the model between size and time that is used. The thirteen methods are:

- Average Estimated LOC (AEL) uses the average rate of development using estimated lines of code for the projects.
- Average Actual LOC (AAL) uses the average rate of development using the actual lines of code for the projects.
- Average Estimated Methods (AEM) uses the average rate of development using estimated number of methods for the projects.

- Average Actual Methods (AAM) uses the average rate of development using the actual number of methods for the projects.
- Linear Estimated LOC (LEL) uses a linear regression model using estimated lines of code for the projects.
- Linear Actual LOC (LAL) uses a linear regression model using the actual lines of code for the projects.
- Linear Estimated Methods (LEM) uses a linear regression model using estimated number of methods for the projects.
- Linear Actual Methods (LAM) uses a linear regression model using the actual number of methods for the projects.
- Exponential Estimated LOC (EEL) uses a exponential regression model using estimated lines of code for the projects.
- Exponential Actual LOC (EAL) uses a exponential regression model using the actual lines of code for the projects.
- Exponential Estimated Methods (EEM) uses a exponential regression model using estimated number of methods for the projects.
- Exponential Actual Methods (EAM) uses a exponential regression model using the actual number of methods for the projects.
- the PSP time estimation method (PSP) uses the flow chart shown in Figure 2.3. Based upon the correlation of the size data in LOC, the user uses either LEL, LAL or AAL.

This empirical investigation required the Leap toolkit's level of automation to make the data collection and analysis possible. Finding a more accurate estimation method may improve software developers' ability to estimate their projects.

#### **4.2.1 Empirical Investigation Environment**

I performed the empirical investigation in a student environment in the ICS 613, Introduction to Reflective Software Engineering, course at the University of Hawaii Manoa. The students in the class developed 8 software projects and recorded their software processes using the Leap

toolkit. By reflecting on their experiences and the data they collected about their software development processes they learned how to improve their development processes. During the development process they estimated the size of and time needed for the last six software projects. The Leap toolkit provided an automated tool for looking at historical development data and deriving effort estimations based upon historical size and effort data. The Leap Time estimation tool provided the students with many different effort estimates based upon their historical data. The student could use these Leap toolkit generated estimates to make their own estimate of how long the project will take. The students recorded their estimate and the Leap toolkit provided me with the other 13 estimates automatically.

#### **4.2.2 Independent and Dependent variables**

Since the objective was to evaluate the different quantitative time estimation methods, the independent variable was the estimation technique. This means that there was one independent variable that can take on 14 different values: the 12 method values, the PSP value and the student's own estimate. There was one dependent variable in this empirical investigation, the relative prediction error. The relative prediction error was calculated for each estimation method for each student and for the entire class. The following dependent variable was calculated for each of the 14 different estimation methods:

$$\text{Relative Prediction Error} = |\text{estimate} - \text{actual}| / \text{actual}$$

#### **4.2.3 The Design**

I divided this empirical investigation into two parts, individual students and the class as a whole. At the beginning of each project each student developed a planned size and planned effort for the project. After the third project, the Leap toolkit estimated the effort according to the different estimation methods (treatment, alt 1 - alt 13). The student produced the 14th estimate. During the project the students recorded the actual amount of effort and size of the project in the Leap toolkit. After all the projects were finished, I calculated the relative prediction error for each of the fourteen estimation methods. I could not use the data from the first three projects since the quantitative estimation methods require at least three data points.

#### 4.2.4 Analysis

I used the following relationship to model the empirical investigation  $y_{ij} = \mu + t_i + e_{ij}$  where:

- $y_{ij}$  = the relative prediction error for alternative i
- $\mu$  = the overall mean
- $t_i$  = the effect of the ith treatment (estimation method)
- $e_{ij}$  = residual for the ith and jth treatment.

I analyzed this model with standard analysis of variance (ANOVA) procedures with the null hypothesis:

$$H_0 : \mu_1 = \mu_2 = \mu_3 = \dots = \mu_{14} \quad \text{where} \quad \mu_i = \mu + t_i; i = \{1, 2, 3, \dots, 14\}.$$

The null hypothesis states that there is no effect of estimation method on the prediction error. I tested the null hypothesis for each student and for the class as a whole.

The next chapter presents the results of this research.

# Chapter 5

## Results

*Don't tell people how to do things. Tell them what to do and let them surprise you with the results.* – George Patton

This chapter presents the results of this research. There are three primary results:

1. The current design of the Leap toolkit, which has evolved in response to user feedback over the past two and a half years.
2. A case study of 16 graduate students using the Leap toolkit to explore the relationship between collection stage errors and the overhead associated with data collection.
3. Using the Leap data that the students recorded, I was able to conduct an empirical investigation into the accuracies of different estimation methods that provides useful new insights into personal estimation and planning methods.

Section 5.1 discusses how I developed the Leap toolkit with the help of the users of the Leap toolkit. Section 5.2 presents evidence that reducing the overhead of data collection to the user reduces the level of collection errors. Section 5.3 shows the differences between different time estimation methods. Section 5.4 presents some interesting results from the case study that provide further insight into software developer improvement. The chapter concludes with an analysis of the main hypothesis (Section 5.5).

### 5.1 Leap's design as a response to user feedback

The development of the Leap toolkit followed a method similar to action research[16]. I was trying to build a tool that would help software developers learn more about themselves and

improve their own software development practices. To build an effective tool, I had to have software engineers use the tool and provide me feedback. Over the past two and a half years there have been many cycles of use and feedback that have provided insight into software engineering and the problems associated with collecting individual software engineering data. The focus of the evolution of the Leap toolkit has always been to provide an efficient, flexible tool that supports the user's software process and helps them become better software engineers. The current design of the Leap toolkit is available as a set of JavaDoc pages from the Leap homepage <http://csdl.ics.hawaii.edu/LEAP/LEAP.html>. The next sections discuss some of the major changes to the Leap toolkit as a result of our experiences.

### **5.1.1 From single to cross-platform availability**

My research group, the Collaborative Software Development Laboratory (CSDL) at the University of Hawaii, Manoa has a tradition of very good research into software engineering issues. Previous software engineering research at CSDL, such as AEN[29], CSRS[38, 40, 39], and CLARE[41] produced powerful tools for collaboration and research. However, these tools required an infrastructure not commonly found. The tools required Unix, C++, and Emacs Lisp. The user interface for all three of these projects was the Emacs editor. While being a very powerful editor, Emacs does not have a user friendly interface for novice users. When we tried to transfer these technologies to industry, we found it very difficult. Not enough developers use Emacs or have Unix boxes to run the database backend for our tools. Another issue with our older systems was all the data was collected in a central database. This raised the issue of measurement dysfunction, since managers had access to all the data collected and the users could not "hide" their data.

In the Spring of 1997 we decided that we wanted actual software engineers to use our tools and gain more benefit from our research. This decision forced us to change our choice of development languages. Since we wanted our tools to run on many different platforms, we choose to implement our new tools in Java. Another change was to focus more on the user interface. Since we want more people to use our tools, the user interface must be simple and easy to use. Again, Java helps with this problem. Java provides a package of graphical user interface components that work on many different platforms.



### 5.1.2 From simple to sophisticated data collection and analysis

The first instance of the Leap toolkit was a pure defect collection/analysis tool for review support. My intent was to make a very simple review support tool, building from our experiences with the CSRS review tool. There were just two tools LeapDet and LeapDat. Figure 5.1 show the LeapDet tool a simple defect entry tool. LeapDet allowed the user to enter defects about a

Figure 5.1. LeapDet. Simple defect entry tool.

workproduct. It was very similar to the PSP's Defect Recording Log. It had a single row for each defect and collected the same information about each defect. The reviewer could email the defect they found to the author and the other members of the review. Figure 5.2 shows the LeapDat tool a simple defect analysis tool. LeapDat provided some very basic analyses of the defect data. After showing LeapDet and LeapDat to some Software Quality Assurance engineers and getting feedback from them, I added some short cuts so that the user could select from defined values so they would not have to type in the same value every time.

Very quickly I realized that I needed to add time and size recording and analysis to the Leap toolkit. So, I added more tables to the Leap toolkit for recording time and size. I also added tables for defining document types, defect types, projects, and phases. Many of the students in the first graduate software engineering class to use the Leap toolkit complained that time data collection was too hard. Dr. Johnson gave them the assignment to develop a better time collection tool. Each



student developed different time collection tools for their own use. One of the tools, developed by Robert Brewer, was of sufficient quality to be integrated into the Leap toolkit.

### **5.1.3 From opaque to translucent data representation**

The early uses of the Leap toolkit showed me another problem with my implementation and design choices. I had decided to store all the data as serialized Java objects. This greatly simplified the program, but lead to some very severe problems. When I upgraded the version of Java, all the users' stored data could no longer be loaded. Some other users were complaining that they did not know what the Leap toolkit was saving and they were not sure of its accuracy.

These two issues forced me to change the way the Leap toolkit stored the data. I designed a new data format involving ASCII storage of data using HTML table representation. This change provided flexibility and visibility. The Leap toolkit could easily parse the data and I could easily add or change columns in the tables. Users could read and even edit their own data. They now know exactly what is being stored about them.

Changing to an ASCII file system also provided more flexibility. The user could store different types of data in different files. I felt this was a great benefit to users. However, when we tried to teach new users how to use this feature they quickly became confused and didn't know where their data was being save or if it was saved. Dr. Johnson and I had many long discussions about the issue of saving data. Eventually, we decided that we would teach new users to use a single data file. The students in the case study used a single data file to store all of their Leap data. As the data file got larger over the semester, the Leap toolkit slowed down while loading this large file. We are now developing a personal database as another storage mechanism to solve this problem.

### **5.1.4 From simple to sophisticated project management**

Another issue that evolved due to user feedback was project management. The first version of the Leap toolkit (LeapDet and LeapDat) had no support for project management. After I added the project table for defining projects, users wanted a mechanism to support project summaries. The initial tool support for project summarization were very rudimentary. It only produced a report similar to the PSP Project Summary report. The students that used it liked the summary, but wanted more. They also wanted a time estimation tool, so I started development of the TimeEstimation tool that would allow users to quickly estimate how long the next project will take. I also

started development on the Hee tool for project summarization. Based upon user feedback to Hee, the next step will be to work on a tool that supports effective project comparison.

The Leap toolkit has evolved to meet the needs of the users. By helping the users become more effective at data collection and analysis, I have improved the Leap Toolkit.

The second result of this research results from an investigation that I conducted on graduate students.

## **5.2 Collection errors and collection overhead**

In the case study, students were asked to collect three forms of data (size, time, and defects) with the resulting possibility for three kinds of collection error. Although the Leap toolkit supports collection of other forms of data (checklists and patterns), these were not considered in the case study.

To reduce collection error in size data, the Leap toolkit uses LOCC to count the size of the projects. We have extensively tested LOCC to ensure that it counts Java programs consistently. LOCC counts the size of Java programs and can determine the amount of new code in programs. It also produces Leap data files. By using LOCC to count the size of their projects, the students get consistent size counts. Another feature of the Leap toolkit that reduces size data collection error is the representation of size data in the Leap toolkit. For Java code, the Leap toolkit counts the number of lines of code for each method in each class in each package. There is a size entry element for each method in the project. For a medium sized project there may be hundreds of size entries for each project. It is very difficult to manually produce all this data, so to enter this data by hand is very difficult.

For the case study, I was not able to measure the level of collection error in the students' defect data. This is because Dr. Johnson taught the students several methods for collecting defect data. Some of the methods were to collect all the defects made, collect only the most important defects, and collect only the most costly defects. He suggested that the students try different defect collection methods to learn which one they like the best. Since the students were not using a consistent defect collection process, I was not able to analyze their defect data to detect collection errors.

The third potential type of collection error is time collection errors. All the students were told to record the time they spent developing their projects. Dr. Johnson showed them the two tools

in the Leap toolkit that support time data collection: Naia and Io. Figures 5.3 and 5.4 show Naia and Io.

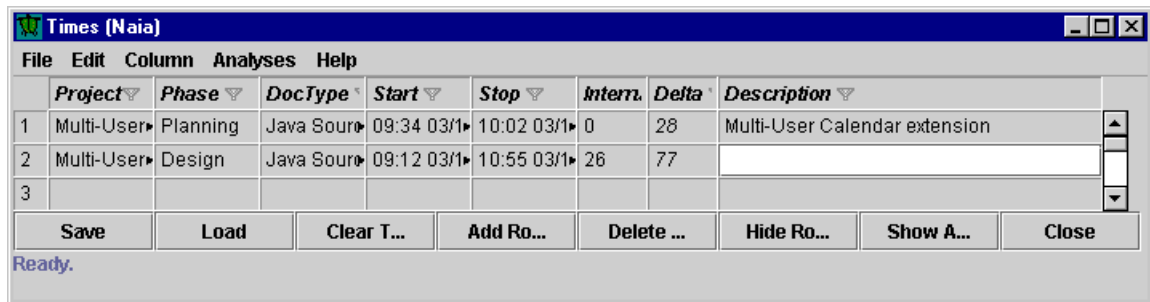


Figure 5.3. Naia time entry/editing tool.

The Naia tool's layout is similar to the Time Recording Log used in the PSP. However, it has the following advantages over paper forms:

- Choice of defined values. Once the Leap user has defined projects, phases and docTypes, Naia presents these defined values to the Leap user. The Leap user can easily select from their defined values. This speeds up data entry and helps ensure that the values entered are consistent and correct.
- Default values from last entry. Once the Leap user has selected a project, phase, and docType, those values are inserted automatically into the next row when the Leap user starts a new time entry. Since most time entries are for the same project, docType and phase as the previous one, this speeds up data entry. The Leap user can easily change the choices that are different.
- Current time entered. When the Leap user starts a new time entry, Naia automatically fills in the start time with the current time. This greatly simplifies data recording, if the Leap user is using Naia to record their work. When the Leap user selects the stop time field, Naia fills in the current time. This also speeds up data entry when the Leap user is recording their work.

However, a problem with this behavior occurs when the Leap user records time data *after* they have completed the work. Since Naia fills in the current time by default, the Leap user must edit the time entries. This may take more time than just typing in the correct time entry. A student who exclusively used Naia to record their time after they finished their project said "I think it's not very convenient when I input the time data into Naia." The student had to edit all the time fields to match the times the student remembered.

- Automatic duration calculation. The Leap user enters the start and stop times and the amount of interrupt time and Naia automatically calculates what the duration is. This is much better than a paper form where the Leap user must do this calculation.

Although Naia provides reasonable support for time entry during project work, we designed a second tool called Io with even more specialized time recording support. Io greatly reduces

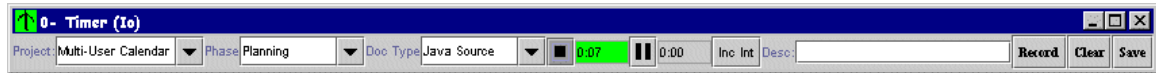


Figure 5.4. Io time recording tool.

the overhead to the Leap user when they are collecting time data during project work. With Io, the Leap user presses the start button when they start working and press the stop button when they are done. If they are interrupted they press the pause button to record interrupt time. Once they have a time entry they press the record button and the entry is added to Naia. Since Io automatically sets the start and stop time, the Leap user cannot use Io to record or edit time data after the fact.

To investigate time data collection error, I analyzed both qualitative and quantitative data on the two tools.

### 5.2.1 Using Naia for data entry makes data entry harder

During the post-class interview, those students who reported that they used Naia indicated that data collection was harder than those students who reported that they used Io. In the interviews, the students, who said they used Naia to record their time, reported a range between neutral and hard in answer to the question of how hard was it to record time data. On the other hand, students who reported using Io to record time said that recording the data was in the range from neutral to easy. One of the students, who had used the PSP, said, “Well, certainly easier than if I was doing it the Watts Humphrey’s way, but still not easy enough.”

Since the students said that using Naia was neutral to hard, I suspected that collection error may be occurring. To investigate this possibility, I analyzed the data to see if Naia-collected data appeared to be “rounded” in some fashion. This would indicate that collection error might be occurring — That Naia-collected time data does not accurately reflect the actual time spent on the project.

### 5.2.2 Reported use of Naia has a higher correlation to rounded times

Students 2, 5, 7, 9, 12, 14, 15, and 16 reported that they used Naia, the time editor tool, to enter some of their time data. Because of the presence of Io, which dramatically simplifies data collection during the project work, students are more likely to use Naia to enter their time data after they have performed the task.

To explore the possibility of inaccurate time data collection, I analyzed the start, stop and duration entries looking for patterns. For all of these analyses, I just looked at the number of minutes. For example, a start time might be 11:30 26 Jan 2000. I only looked at the minutes field, 30. I conducted two different pattern checks on the time data.

First, I looked for times that were a multiple of five. I calculated the percentage of times that were a multiple of five compared to all time entries. Figure 5.5 shows a graph of all the students' data. The average for the class was 28.79%. Now, if the students use Io to record their time and they randomly choose a starting time then, on average, 20% of the start and stop times would contain multiples of 5. However, four students, numbers 3, 7, 14 and 16, had averages that were larger than

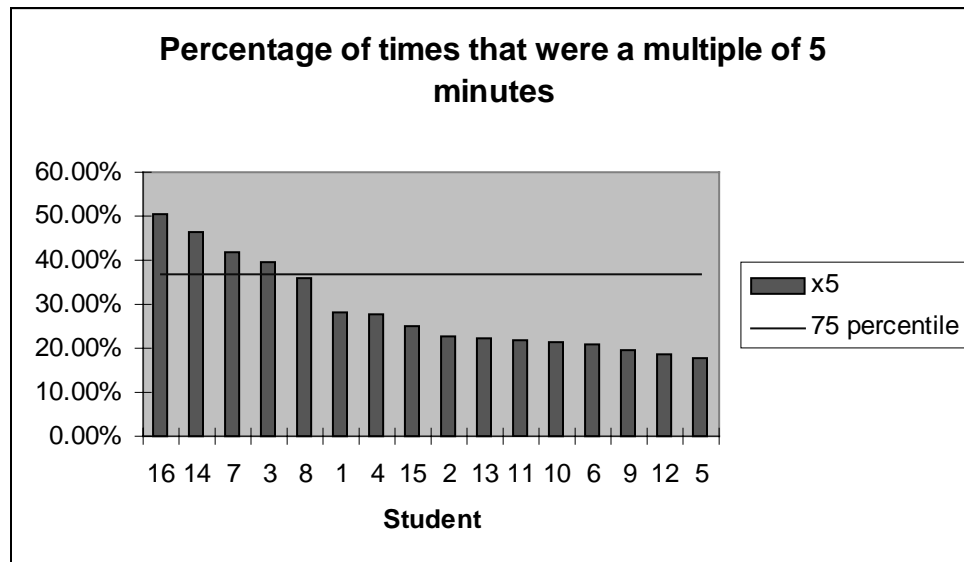


Figure 5.5. Students' percentage of time entries that are a multiple of 5.

the 75 percentile of the class.. Their averages were 39.6%, 41.8%, 46.2%, and 50.5%.

Second, I looked for start and stop times that were different by multiples of 60 minutes. For example, a start time of 11:48 in conjunction with a stop time of 12:48. For this pattern, I expanded the matches to include start and stop times whose minutes differed by one minute. For

example, a start time of 11:48 and a stop time of 12:49. I included this pattern because of the way Naia allows users to enter time. Naia automatically fills in the current time when the user clicks into the start or stop cell. If the user is entering their data after they have completed the task, it is easier to change just the hours and not the minutes. I also believe that it is rare for people to work for exactly multiples of 60 minutes. Figure 5.6 shows the results of this analysis. Four students, numbers 3,

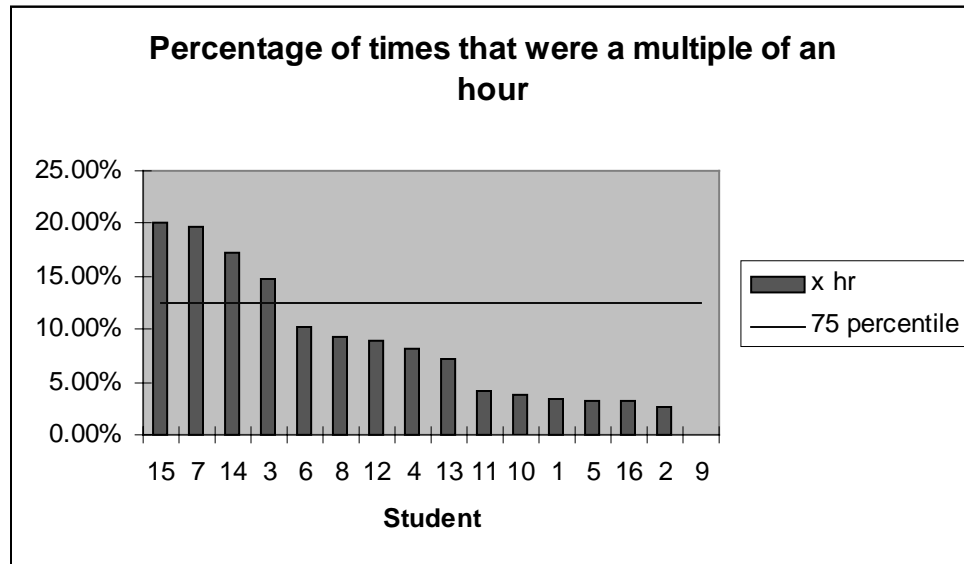


Figure 5.6. Students' percentage of start and stop time entries that the same or different by 1 minute.

7, 14, and 15, had percentages higher than the 75 percentile for the class. Their percentage values were 14.7, 19.6, 17.1, and 20.0.

Combining the results of these two analyses, it looks like students 3, 7, and 14 have patterns in their time data that suggest that they were rounding their time data.

This suggests that their data does not accurately reflect their actual development practices. Interestingly, one of the three students with suspicious time data, 3, said that they preferred using Io to record their time data. Students 7 and 14 had higher percentages than did student 3. This might indicate that student 3's data might be fairly accurate.

### 5.2.3 Limitations

A first limitation of this research is that I conducted a case study instead of a controlled experiment: I did not use randomized controls or a control group to compare against my experimental group. Because I used the case study methodology, I did not attempt to derive statistically significant conclusions concerning the relationship between collection overhead and errors. I used



the case study methodology since software engineers do not develop their software in a controlled environment. The case study environment was closer to the environment that the students will be working in when they are programming professionally.

A second limitation is that the subjects of this case study were all graduate students in computer sciences. They do not accurately represent software engineers in industry. Half of the students had less than two years of professional software development experience. The students had different time constraints than real software engineers and their motivation to produce high quality software efficiently is very different than the professional software engineer. For example, the students were not going to be fired if they produced poor quality software.

Another limitation of this case study is the sample size. I had access to only 16 students. All of the students came from the same university. I cannot make generalizations about all software engineers from a sample size of 16.

#### **5.2.4 Summary**

The results of the case study support the claim that the reduced collection overhead reduces collection errors for the following reasons. First, the time entry tools in the Leap toolkit are easier to use than paper forms. Naia has less overhead than paper forms because it provides default values for most of the fields, it automatically fills in the start and stop time when the user clicks in them, and it automatically calculates the duration. Io has even less overhead. The user just has to click the start and stop button to record time. Second, the data analysis showed that the students who used Naia had data that looked like it was more rounded and did not accurately reflect their development practices. It appears that higher overhead correlates with collection error.

The third result of this research results from an empirical investigation into the accuracies of different time estimation methods.

### **5.3 Comparing time estimation methods**

The ability to accurately estimate the effort for a project may depend on the estimation method. Since the Leap toolkit supports many different methods of time estimation, I was able to conduct an empirical investigation to determine if any individual student had a better estimation method and if there was a better estimation method for the entire class.

In the rest of the charts in this dissertation, I use simple box and whisker plots. The box and whisker plot shows the minimum, 25 percentile, median, 75 percentile, and maximum values for the data. Figure 5.7 gives an example of a box and whisker plot.

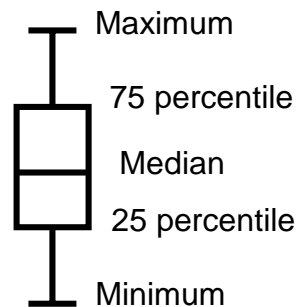


Figure 5.7. A simple Box and Whisker plot.

### 5.3.1 Size estimation skills improved

I looked at the students' size estimation skills at the LOC and method level. Figures 5.8 and 5.9 show the distribution of the size estimation accuracies for LOC and methods. The students'

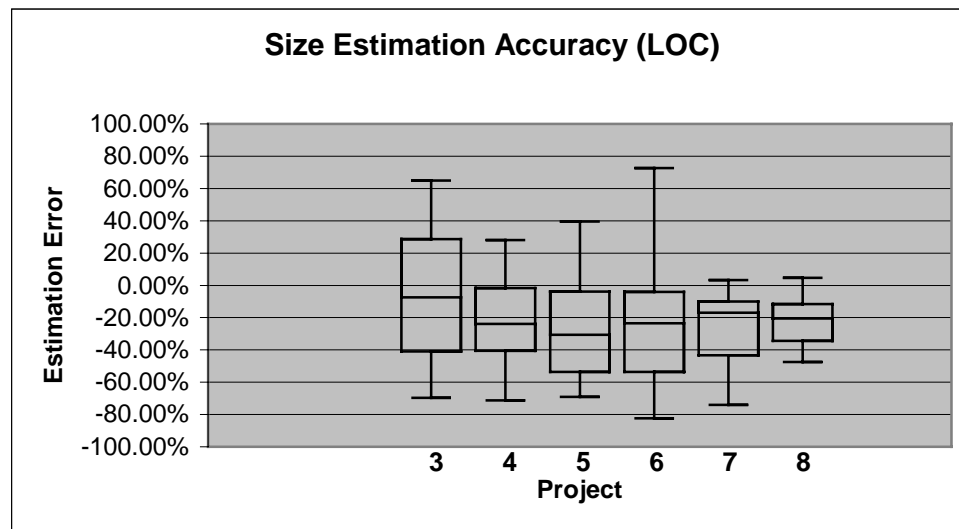


Figure 5.8. Size estimation accuracy (LOC). The class' average accuracy leveled off. The class tended to underestimate how large the projects would be.

average LOC estimation error did not improve; it decreased slightly. As a whole the class became more consistent in their estimation. The range of estimates became smaller overtime, but they tended to underestimate the number of lines of code their projects would take. This is probably

because of the method the students were using to predict the number LOC. First, The students designed the project and determined the number of methods it would take. Then, they determined their historical average number of LOC per method. Then they multiplied the number of methods by the LOC/Method to determine the number of LOC for the project. Since the average number of LOC/Method increased the students were underestimating the number of LOC in their projects.

Figure 5.9 shows that the students' average estimation error for methods rose from -50% to -14%. This could imply that the students' design skills were improving. They were better able to determine what methods were needed before they started coding the projects.

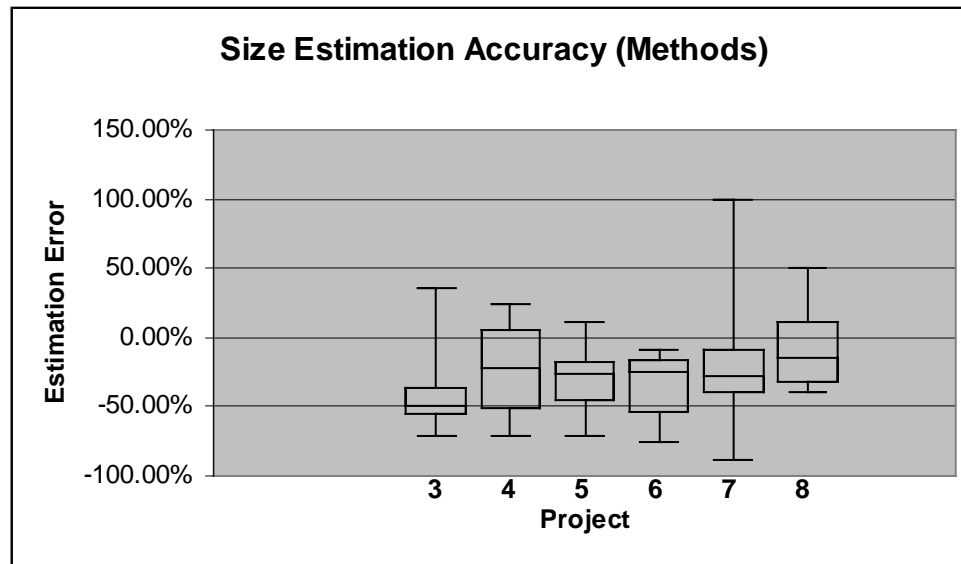


Figure 5.9. Size estimation accuracy (Methods). The class' average accuracy increases over time. The class tended to underestimate how large the projects would be.

### 5.3.2 Time estimation skills improved

The time estimation skills of the students did improve. For the first four projects on average they tended to underestimate their project time by about 17%. Over the last two projects on average they improved their time estimation to slightly over-estimating the projects.

I compared the correlations between the different size and time estimate accuracies. Table 5.1 shows the results. These results imply that as the students got better at design, their time estimates also got better. Improved design results in a better estimate of the number of methods, but not necessarily better LOC estimates. These results are very similar to published PSP results[14]. I

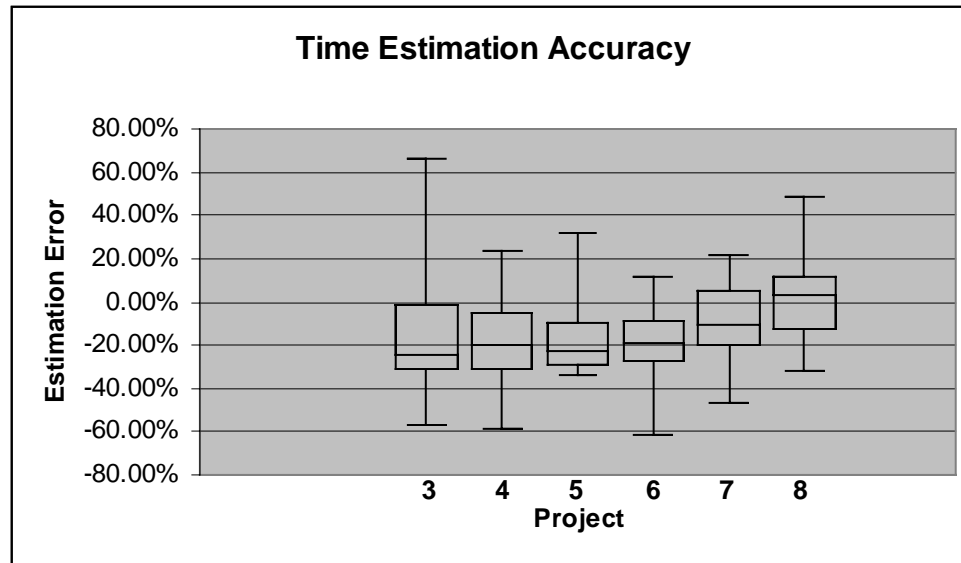


Figure 5.10. Time estimation accuracy. The class' average accuracy increases over time. The class tended to underestimate how long the projects would take.

Table 5.1. Correlation between Time, LOC, and Method estimation accuracy

	<i>Time</i>	<i>LOC</i>	<i>Methods</i>
Time	1		
LOC	-0.26	1	
Methods	0.84	-0.48	1

wanted to see if there was a difference in estimation methods, so I conducted an empirical investigation.

### 5.3.3 Empirical Investigation

After collecting all of the students' Leap data, I wrote a data analysis tool that took each of the student's data files and calculated the 13 different estimates for each of the last 5 projects. The Leap toolkit's architecture and design allowed me to design and build this analysis tool in less than three hours. The analysis tool calculated the relative prediction error for each of the different estimation methods and the student's own estimates. The relative prediction error is defined as:

$$\text{Relative Prediction Error} = |\text{estimate} - \text{actual}| / \text{actual}$$

After calculating the relative prediction error for all the estimates the analysis tool performed an ANOVA calculation to determine if there was a significant difference between the relative prediction errors of the different estimates.

The empirical investigation had two parts. I analyzed each individual student's data to determine if there was a best estimation method for the student. Then, I combined all the students' data to determine if there was a best estimation method for the class as a whole. The next section discusses the individual student results.

#### 5.3.3.1 Individual Student Results

Three out of the 16 students had an estimation method that was a significantly better time estimation methods than all others. For the other 13 students there was no significant difference between the estimation methods.

For two students, #s 12 and 15, their own estimation skills were significantly better than any of the other empirical methods ( $F = 3.93$ ,  $P < 0.001$ ) and ( $F = 2.25$ ,  $P = 0.019$ ). Tables 5.2 and 5.3 show the relative prediction errors for the two students.

For the third student, # 4, the exponential regression using actual LOC method was significantly better ( $F = 2.34$ ,  $P = 0.014$ ). Student #4's own estimation skill was the next most accurate method. Table 5.4 shows the relative prediction error for student # 4.

Table 5.2. Student #12's relative prediction error

Estimation	Average			
Method	Count	Sum	Error	Variance
Student12	5	70.5	14.1	116.375
EAL	5	122.4	24.48	239.457
EAM	5	125.7	25.14	250.678
AEM	5	135.2	27.04	160.573
AAM	5	137.8	27.56	127.333
AEL	5	138.6	27.72	199.737
LAL	5	141.8	28.36	110.303
AAL	5	142.5	28.50	171.535
LAM	5	147.0	29.40	80.54
PSP	5	148.9	29.78	130.977
EEM	5	213.9	42.78	115.437
EEL	5	217.2	43.44	154.453
LEM	5	319.6	63.92	909.772
LEL	5	323.9	64.78	1072.447

Table 5.3. Student #15's relative prediction error

Estimation	Average			
Method	Count	Sum	Error	Variance
Student15	5	45.8	9.16	59.628
EAM	5	61.2	12.24	89.468
LAM	5	62.3	12.46	71.668
EAL	5	69.8	13.96	73.283
LAL	5	71.5	14.30	96.415
EEM	5	130.1	26.02	148.402
EEL	5	134.6	26.92	117.312
LEM	5	148.2	29.64	188.678
LEL	5	154.3	30.86	139.428
AAM	5	211.0	42.20	627.735
AEM	5	249.3	49.86	967.753
PSP	5	260.2	52.04	2085.193
AAL	5	260.2	52.04	2085.193
AEL	5	294.2	58.84	2656.443

Table 5.4. Student #4's relative prediction error

Estimation		Average		
Method	Count	Sum	Error	Variance
EAL	5	92.6	18.52	333.002
Student #4	5	94.0	18.80	450.415
AEM	5	109.1	21.82	721.077
AAL	5	135.2	27.04	477.523
PSP	5	136.8	27.36	477.203
EAM	5	163.2	32.64	365.933
LAL	5	215.8	43.16	594.073
AEL	5	217.8	43.56	893.478
AAM	5	226.8	45.36	293.833
EEL	5	330.4	66.08	1712.512
EEM	5	337.1	67.42	1127.122
LAM	5	385.2	77.04	4812.388
LEL	5	393.1	78.62	2857.387
LEM	5	419.9	83.98	1679.997

### 5.3.3.2 Class Results

When I analyzed the entire class' Leap data, I found that there was a significant difference in the relative prediction error ( $F = 3.75$ ,  $P < 0.001$ ). Table 5.5 shows the relative prediction error for the entire class.

When looking at the entire class data set, the students' estimates were better than any of the "mechanical" methods including the method used in the PSP. Perhaps, this is because the students were able to consider external factors such as how many new features they would have to learn, or how similar the current project was to previous projects when they were making their estimates. The students also had the opportunity to view any of the empirical estimates before they made their estimates. This result is not very surprising. It seems very logical that the students would be able to estimate their amount of time better than any single method.

When I removed the students' estimates from the data, I found that there was a significant difference between the mechanical methods ( $F = 2.40$ ,  $P = 0.005$ ). Among these remaining methods, the exponential regression model using actual methods is significantly more accurate than the rest of the methods. This makes some intuitive sense. As the number of methods increases, the number of interactions between the methods increase, but not in a linear manner. More interactions leads to more complexity that the developer must control. Controlling this complexity requires more time for larger projects.

Table 5.5. Relative Prediction Error

## SUMMARY

Method	Average	Variance
Student:	19.77	175.86
EAM:	27.92	370.26
EAL:	29.52	1663.59
LAL:	30.88	780.50
AAL:	34.74	508.23
PSP:	34.9	508.89
AAM:	38.88	426.94
EEL:	39.57	1590.45
AEM:	40.99	1420.99
EEM:	41.70	1607.55
LAM:	42.69	10621.90
AEL:	46.7	1724.62
LEL:	53.45	3423.73
LEM:	55.02	3389.65

### 5.3.4 Limitations

As with the case study, a major limitation with the empirical investigation is the sample size. Eight programs is a relatively small number on which to base estimation. Since they wrote so few programs they had relatively few data points for the estimation methods. In addition, A high percentage of the program estimates were near the end points of the size range. Regression analyses do not work well near the endpoints. If the students had more data points then the estimation error for the methods might have been different.

### 5.3.5 Summary

For one student, the Leap toolkit provided a more accurate method of time estimation than the student actually used. If the student knew this, they would have been more accurate at their time estimation. They could have looked at the EAL estimate and then adjusted their own estimate based upon what they knew about the project.

The class' results show that the students are generally better at estimating than any of the "mechanical" methods. Since the students saw the estimates that the Leap toolkit generated, it makes sense that the students were better estimators. When I compared just the 13 mechanical estimation methods, the exponential regression using actual methods was most accurate. This does not mean that all software developers should be using the exponential regression with actual methods



as their time estimation method. What it might imply is that if you are estimating the time a large project with many developers may take, then you might want to use the exponential model with actual methods, since it appears more accurate for groups of developers.

## 5.4 Additional Results

In addition to the above results, I found some other interesting results not directly related to my central claims about the Leap toolkit. This section discusses some of the results that I found from the case study.

### 5.4.1 Rounding errors did not affect the accuracy of the estimates

Student 14 reported using Naia for time entry and has data that appears to be rounded. During the interview they said that they used Naia exclusively. They said “Normally I would record my time after I finish and then I would record the time according to my remembrance.” This indicates that the student’s data may not accurately reflect what really happened. Certainly, it is not as accurate as recording the time while they are programming. After finding this, I attempted to determine if this decidedly inaccurate approach to time entry had any effect on the student’s estimation accuracy.

I set up a very simple hypothesis: that collection error does affect the accuracy of the students’ estimate. To test this hypothesis, I divided the students into two groups, students with suspicious time data (i.e. data that appears to be rounded) and students without suspicious time data. I calculated the student’s estimation accuracy ( $|\text{estimated time} - \text{actual time}| / (\text{actual time})$ ) for each project. I did an ANOVA test to see if there was a significant difference between the groups estimation accuracies. Figure 5.6 shows the results of the ANOVA analysis. Interestingly,

Table 5.6. Class’ estimation error

Summary						
Groups	Count	Sum	Error	Variance		
Suspicious data	40	676.6	16.92	88.74		
Not Suspicious	40	903.3	22.58	251.03		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	642.41	1	642.41	3.78	0.06	3.96
Within Groups	13250.87	78	169.88			
Total	13893.28	79				

there was no significant difference between the estimation error of the groups. Therefore, I cannot reject the null hypothesis and my hypothesis that collection error affects the estimation accuracy is not supported.

Thus, the level of collection error I observed does not appear to affect the accuracy of the students' estimation abilities. Interview data suggests that the students did not feel that collection error caused a problem with estimation. In the interviews six students felt that estimation was one of the most important thing that they learned from the class. Even with their collection errors, they felt that they were able to effectively estimate their projects. Most of the students were comfortable with their estimates and were able to accurately estimate how long their next project would take.

#### **5.4.2 There was very little evidence of measurement dysfunction in the class**

Measurement dysfunction is the situation where the act of measuring performance causes the people being measured to increase the measures while actually reducing the quality of their performance.

Two factors that can contribute to measurement dysfunction in a classroom setting are grading pressure and class competitiveness. There was no evidence of either of these pressures during the case study.

Dr. Johnson reduced the first pressure by telling the students that he would not grade the students' estimating accuracy. To further reduce the pressure on the students, he did not show the students the Time Planning Accuracy chart until the last two projects. One student said, "He [Dr. Johnson] said in the beginning clearly, he says [estimating accuracy] does not matter that it is not an issue."

The students in the class did not share their Leap data with each other. All students interviewed said they did not compare their Leap data with the other students in the class. This is very different from previous classes I observed, where the students did compare their results. I conjecture that this may be because the students did not make paper copies of their results. Instead, they turned in their data on floppy disks. Then, Professor Johnson loaded their data into Leap to review it with each student individually. Not having printouts made it much harder for students to compare their data with each other. With printed charts and reports the students can easily compare their data before class starts. With their data on floppy disks and without computers in the classroom, the students could not easily compare their results before class.

Most of the students reported that they felt little pressure to make their data look "good". In the surveys, seven students reported decreased levels of pressure to make their actual times match

their estimated times during the semester. Two students agreed more strongly at the end of the course than at the beginning of the course that there was pressure to make their time estimates match the actual times. Four students remained consistent at low levels. This implies that the students in general did not feel very much pressure to make their actual time match their estimates. With less pressure, there is less chance for measurement dysfunction. The two students who reported increasing pressure also reported that they felt very little pressure to turn in “good” data. On the final survey, they still disagreed that there was pressure to turn in matching data.

These results do not imply that the students were not aware of the measurement dysfunction issue. Several students were aware that in the *real world* there might be more pressure to turn in good data. One student said they felt

“not too much pressure, but ... we are students. [I]n the real world ... I think I will have some pressure if I made a bad estimate. I will try to work hard.”

Another student said that making and using estimates “terrifies” them.

“Well, using it for Philip’s class doesn’t terrify me, but, when I used to do this for a living and when I’m going to do it for a living again it just it make me very anxious ... If you get it wrong, that’s really bad.”

These students realized that when they are employed, they may be evaluated by their management based upon their Leap data.

### **5.4.3 Student productivity stayed steady**

The productivity of the students in the class did not change very much, but the class’ average productivity did increase. Figure 5.11 shows the average productivity of the the class in LOC/hr. The first four projects were command line projects. They built upon each other. In project five, the students had to implement a GUI interface for the program. This task was new for them and it is natural for their productivity to decrease, since they have to learn about GUI components in Java. The last three projects were extensions to their GUI. The increased productivity could be a result of the students’ increasing familiarity with Java GUI components.

The students’ average productivity as measured by Methods/hr stayed steady. Figure 5.12 shows the average productivity of the the class in Methods/hr. Combining the two productivities implies that the students’ methods were growing in size. The average LOC/method increased from 11.28 in project 1 to 17.07 in project 8. Figure 5.13 shows the average LOC/Method for the class. This may just be an artifact of the changing nature of the projects. The first few projects did not have a GUI, while the last projects focussed on the GUI. One student said

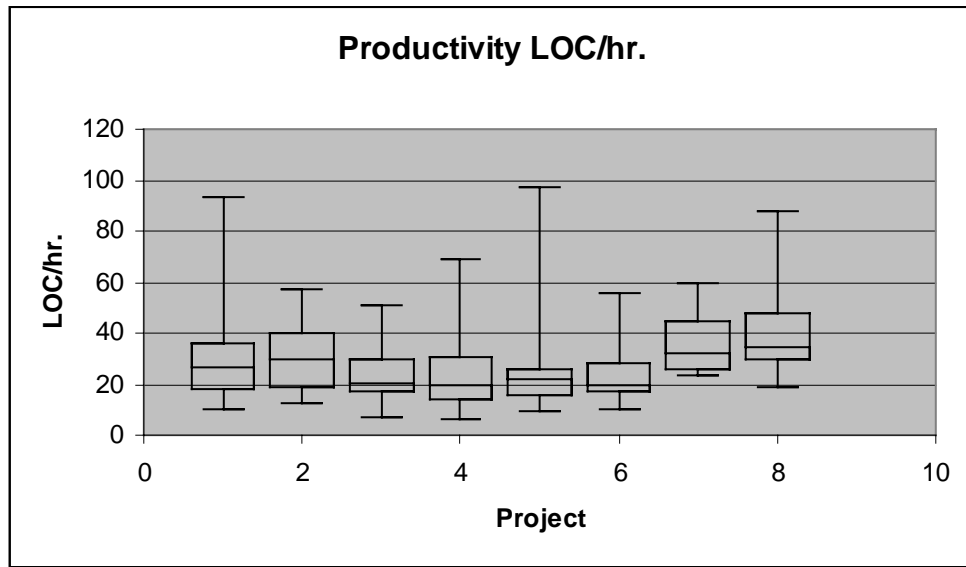


Figure 5.11. Class productivity in LOC/hr. The class average increases slightly over the last 3 projects.

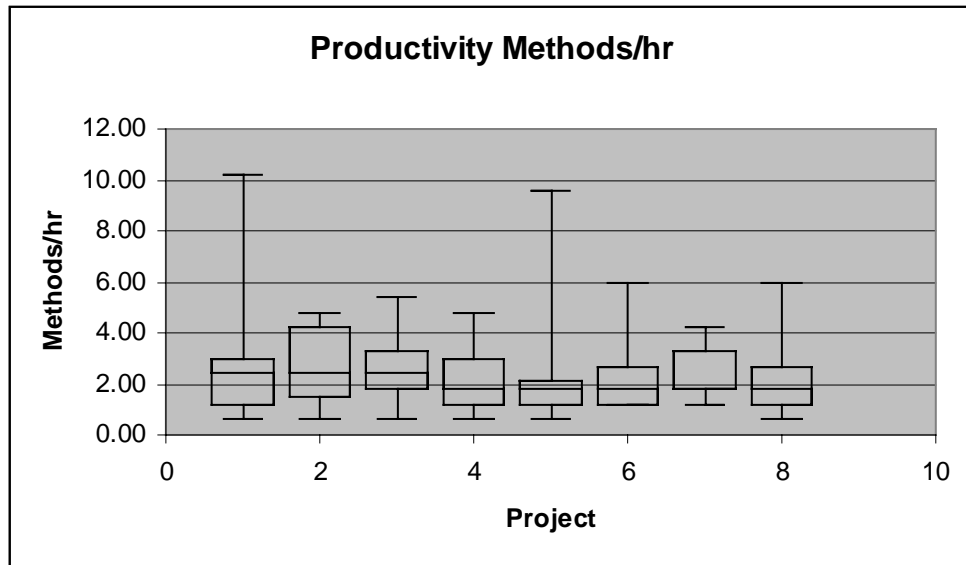


Figure 5.12. Class productivity in Methods/hr. The class average stays roughly the same over the last 3 projects.

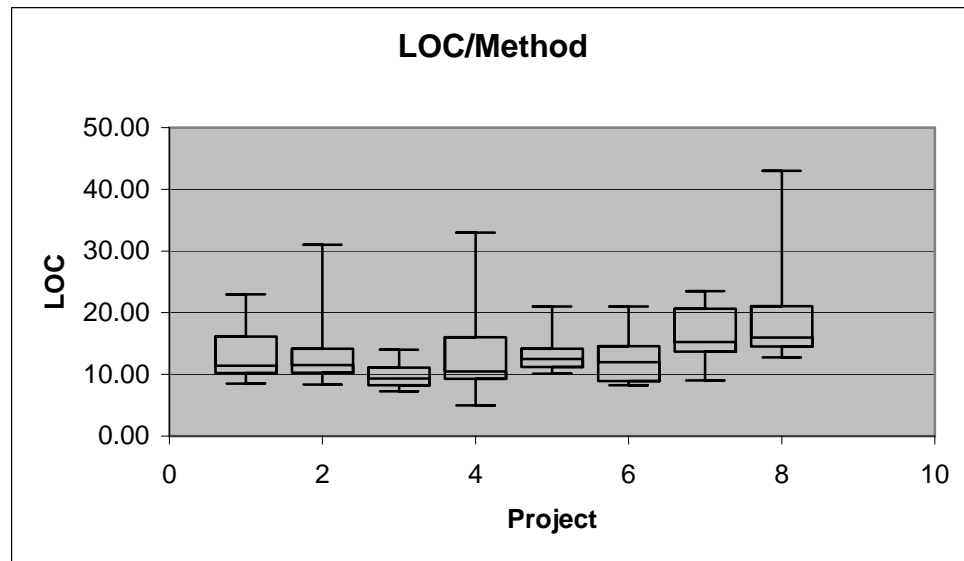


Figure 5.13. Average LOC/Method. The class average increases over time.

The GUI has quite a bit more code than the others. I guess that is what I think. So, I tried to raise the lines of code per method. I tried to adjust that.

This student notice a trend that occurred for the entire class.

#### 5.4.4 Experience leads to overconfidence

Looking at the estimation accuracy data for the students in the class, I noticed that there was a difference between the estimation accuracy for students who reported more programming experience and students with little programming experience. I divided the class into two groups, students who reported less than five years of programming experience and students with at least five years of programming experience. I then compared their size and time estimation accuracies.

First, I looked at the students' size estimation accuracies. The experienced students, on average, underestimated their program size in LOC by 32.5%. The less experienced students underestimated their program size in LOC by only 8%. I found a significant difference between the two groups ( $F = 11.3$ ,  $P < 0.01$ ). When I looked at the students' method estimation accuracies, there were no significant differences between the two groups ( $F = 0.04$ ,  $P = 0.85$ ).

Second, I compared their time estimation accuracies. The experienced students had an average estimation error of -17%. The less experienced students had an average estimation error of

only -7%. I found that there was, again, a significant difference between the two groups ( $F = 3.94$ ,  $P = 0.05$ ).

It appears that the more experienced students tended to underestimate the size and effort required for the programming assignment. This could be due to overconfidence in their programming skills and disbelieving the data that the Leap toolkit was giving them. By the eighth project, however, the experienced students were as accurate as the less experienced students indicating that they eventually began to “trust” their empirical data.

## 5.5 Summary

Since the Leap toolkit reduces the overhead of data collection, and since high overhead in data collection correlates to rounding error, it appears that the Leap toolkit reduces collection errors as compared to manual data collection methods. However, the Leap toolkit does not totally eliminate collection errors and there is room for further reducing the overhead of data collection.

By providing a sophisticated size representation, tool support for counting size, and different estimation models, the Leap toolkit allows users to view many different time estimates for their projects. Based upon these estimates, the users can accurately estimate the time required for their future projects. Without these features users could not effectively compare different estimation methods.

The next chapter discusses some of the implications of these results and future directions for this research.

## Chapter 6

# Conclusion

*The world is round and the place which may seem like the end may also be only the beginning.* – Ivy Baker Priest

The goals of this research were three-fold. First, to provide a design philosophy that addresses some of the drawbacks with traditional software development improvement efforts. Second, to implement a personal software developer improvement tool that follows the design philosophy. Third, to conduct a case study to determine the effectiveness of the Leap toolkit.

This chapter first summarizes how these goals were achieved in this research. It then presents the major contributions of this research to software engineering in general and individual software developer improvement in particular. Next, it discusses the future directions of this research. Finally, it presents some implications for current and future software development improvement practice.

### 6.1 Research summary

After using the PSP, FTR, and other software development improvement efforts for several years, I noticed that while these techniques show many positive results they also suffer from some typical problems.

- Many improvement initiatives impose heavyweight development process constraints on the developers, such as following a strict waterfall development process. The developers must follow the development process to see any process improvement benefits.

- Most traditional software process improvement efforts focus on improving the development organization. Individual developers may indirectly see the benefits of the organization's improvement, but providing benefit to individual developers is not the focus of the improvement effort.
- Many improvement efforts do not address measurement dysfunction. The data collected is not protected to ensure that management will not use it for evaluation of the developers.
- Many software improvement efforts use manual data gathering. Manual data collection and reporting significantly increases the developers workload and often interrupts their work.

To address these issues, I helped develop the LEAP design philosophy. The LEAP design philosophy consists of four principles:

- **Light-Weight** The first principle is that any tool or process used in software developer improvement should be light-weight.
- **Empirical** The second principle is that the methods for software developer improvement should be empirical in nature.
- **Anti-measurement dysfunction** The third principle is that methods for developer improvement should address the issue of measurement dysfunction.
- **Portable** The fourth principle is that any tool or process used in software developer improvement should be portable.

The LEAP principles try to ensure that the benefits of software development improvement efforts are seen by the individual software developers.

In Fall, 1996, I started developing the Leap toolkit. The Leap toolkit is a reference implementation of the LEAP design philosophy. I wanted to find out what the practical implications of the LEAP design philosophy were. After over 25 public releases, the Leap toolkit is over of 44,000 lines of code, 2,209 methods, 287 classes, and 14 packages. The Leap toolkit is available for download from <http://csdl.ics.hawaii.edu/Tools/LEAP/LEAP.html>. The architecture, design and implementation of the Leap toolkit eliminates or reduces many of the data errors that Anne Disney found. By eliminating or reducing these sources of error, the Leap toolkit is more accurate than manual software developer improvement methods.

After using and improving the Leap toolkit for two years, I conducted a case study to confirm that the Leap toolkit is a more accurate and effective way for developers to collect and



analyze their software engineering data than methods designed for manual enactment. The subjects of the case study were the 16 graduate students in the Fall, 1999 ICS-613 (Reflective Software Engineering) class. During the course, I conducted four surveys to find out how the students were using and felt about the Leap toolkit. At the end of the class I interviewed 11 of the 16 students. I also collected all the students' Leap data files.

In the case study, I found that the Leap toolkit reduces the overhead of data collection and that high overhead in data collection correlates to rounding error. The Leap toolkit's sophisticated size representation, tool support for counting size, and different estimation models allowed me to compare 13 different time estimation methods to determine which method was most accurate for each student and the class. I determined that, for the class, the best estimation technique was using an exponential regression on the actual number of methods.

## 6.2 Research contributions

There are three main contributions from this research corresponding to each goal of the research: the LEAP design philosophy for software development processes and tools, the Leap toolkit itself, and the insights gained from the case study.

### 6.2.1 LEAP design philosophy

The LEAP design philosophy addresses many of the issues that I see in current software process improvement efforts.

**Light-Weight** methods and tools reduce the developer's overhead. Reducing the overhead of process improvement allows the developer to focus on the development effort. Light-Weight methods also give the developer flexibility to focus their improvement efforts on areas that they are interested in. They can focus on their estimation skills and not collect defects or they could focus on their defect data and not collect time data.

**Empirical** data collection allows the developer to see quantifiable evidence of their development process. They are able to make judgments and predictions based upon hard data.

**Anit-Measurement dysfunctional** methods and tools help keep the data accurate. If the developer thinks their data will be used to evaluate them, they will feel pressure to make their data look good to the evaluator. In accurate data leads to changes in their development practice which may reduce their effectiveness.

**Portable** methods and tools allow the developer to continue their process improvement efforts even when they change organizations. Developers do not have to restart their improvement efforts each time they change organizations.

The LEAP design philosophy changes the focus of software development improvement efforts from the organization to the individual developer. The basis for any software development improvement effort is the individual software developer. The LEAP philosophy also allows the developer to focus their improvement efforts on the areas they need to improve. The developer should directly benefit from the improvement effort.

### **6.2.2 Leap toolkit**

The Leap toolkit provides benefits for three groups of people: software developers, software engineering educators, and software engineering researchers. First, for the software developer, it provides a powerful set of tools to help them improve their own software development practice. The Leap toolkit's automated support for data collection and analysis helps eliminate many of the data quality issues found in Anne Disney's case study. It also lowers the overhead of data collection and analysis for the developer thus, reducing the level of collection errors. No one I have talked to who has used the PSP then used the Leap toolkit would ever go back to using the PSP.

Second, for software engineering educators, the Leap toolkit allows the educator to show students the benefits of using software engineering principles in their own projects. For example, the students can see a dramatic decrease in the number of defects that make it to testing after they have started using design and code reviews.

Third, the Leap toolkit provides a valuable resource for researchers in software development. The rich data collection and analyses tools provided by the Leap toolkit allow researchers to study how developers build software. Researchers can easily compare different development methods or improvement efforts if the subjects collect their data using the Leap toolkit.

### **6.2.3 Case study insights**

The case study in this research revealed many interesting insights:

- Lowering the overhead of data collection tends to reduce the collection error. The students reported that using Naia for data collection was harder than using Io. In addition, the students who used Naia instead of Io had time data that looked "rounded" and less accurate than the students who used Io.

- The rounding errors that I found did not affect the accuracy of the students' time estimates. This might imply that super accurate data collection is not necessary for reasonably accurate time estimation.
- Measurement dysfunction can be reduced in the classroom by addressing the issue of evaluation and peer pressure. In the Fall 1999, ICS 613 class, Dr. Johnson made it clear that he was not grading the students on their estimation accuracy. In addition, the way he had the students turn in their data made it very difficult for them to compare their results.
- Initially, experience leads to overconfidence. The students in the class who had more programming experience tended to underestimate how long the projects would take. By the end of the class, they were as accurate as the rest of the class.
- For some students, there is a significantly better time estimation method. Two students were significantly better at estimating their time than any of the "mechanical" time estimation methods. For a third student, the exponential regression using actual methods was the most accurate time estimation method.

## 6.3 Future directions

There are many future directions for this research. I categorize them into two groups; exploration of the Leap philosophy and investigations into software developer improvement. The explorations of the Leap philosophy include automated data collection, agents for data analysis, a new data storage system, and new data analyses. The investigations into software developer improvement include replications of the case study and investigating FTR and personal data collection.

### 6.3.1 Automated data collection

Many students said that using Io was good but not good enough. Some wanted the Leap toolkit to automatically know what they were doing and record the time while they were working. I see two different ways of providing automated data collection; make the user's development environment aware and use agents to collect the data for the user.

The first method, making the user's development environment aware, requires that we modify the user's Integrated Development Environment (IDE). Several years ago my research group CSDL <<http://csdl.ics.hawaii.edu>> developed an automated time collection tool, *ActivityLog*, for Emacs. *ActivityLog* knew when you were busy or idle and generated a log of your

busy minutes. Since we implemented *ActivityLog* in Emacs, it knew what file you were editing, the mode of the editor, and the name of the buffer you were editing. With all this information, we could build filters to produce reports with project and task data. *ActivityLog* allowed us to track how we spent our time. We stopped using *ActivityLog* because we did not know how to use the reports to improve our work.

Now, I think future IDEs could incorporate data recording ideas. They could record the time spent working, the size of the workproducts, and the defects that are found during development. Most of today's IDEs already understand many of these concepts: they have projects, they know what files are associated with the projects, they can parse the source code, they can detect some defects and they know when the user is working. Most of this data collection could be done automatically. The IDE could present the user with feedback on their development while they were developing the software. If the user supplied a plan for the project, the IDE could show the user their progress. It could show the current productivity, defect density, or defect detection rate.

The second method for automatic data recording is to provide agents that do the data recording for you. These agents would not be a part of the IDE, but they would observe your work and record the data for you. This method is more flexible than changing the IDE since we could use one agent for many different IDEs, but the agents will not have as much information as an IDE does. The agent method requires a little more overhead by the user. They have to start the agents, tell them what tools they are using, where the program files are stored and where the data should be stored.

There are several interesting issues with all this automated data collection; can we automatically collect accurate data? How can we reduce the overhead to the user of editing the collected data so it is accurate? Will all this data collection introduce measurement dysfunction? What are the privacy issues when we collect all this data? How will the developers feel about all this data collection? Do the benefits of automatic data collection outweigh the costs?

### **6.3.2 Agents for just in time analysis**

Another direction for this research is agents. Agents could perform just in time data analysis. They could look at your Leap data every night and tell you interesting things that they noticed. They could tell you when your productivity is outside your normal bounds and help you reflect on what has changed in your environment. They could track your progress against your estimates and give you warnings about running over or under budget. Agents could send you email messages when they notice interesting trends in your data that you would not think of. The agents

could be additional development coaches that suggest checklist or patterns for use with particular problems or designs.

As with automatic data collection there are many issues with agents. Will agents reduce the overhead to the user? Will these agents introduce more measurement dysfunction? What are the privacy issues when we allow agents to view the user's data? How will the developers feel about agents analyzing their data? Do the benefits of agents outweigh the costs?

### **6.3.3 Database for storing the Leap data**

A direction for this research that we are actively pursuing is using a database to store the user's data. If the Leap toolkit uses a database to store the data then the user does not have to manage the data files. Managing the data files is very confusing for new users, but having control over their own files reduces the threat of measurement dysfunction.

There are many issues with using a database where the user cannot see their data instead of ASCII files that I want to explore. Does this greatly reduce the effectiveness of the the Leap toolkit? Will the users trust the database over ASCII files? What are the issues of measurement dysfunction when you use a database? Can we restrict access to the database? Do users even care about these issues? Future studies with the Leap toolkit can help answer many of these questions.

### **6.3.4 Additional analyses and features**

Currently, the Leap toolkit has a limited set of analyses. I only implemented the analyses that we needed to support our own needs. Since the Leap toolkit collects many different types of data, there are many different analyses that it could perform. Some of these analyses could prove to be very useful. Some of these analyses are checklist and pattern generation, different size trend analyses, and additional time data analyses. The Leap toolkit has very rudimentary defect analyses. I would like to explore the issues in checklist generation from defect data. Can we generate likely checklist items from the defects that are collected?

Another analysis that interests me is pattern generation and detection. Can we look at the defect data and suggest patterns to help solve these defects? Patterns have the potential to greatly improve software development; how can we improve the Leap toolkit's use of patterns?

Another area where the Leap toolkit's data analyses could be improved is in time analyses. There is very little support for project tracking or task breakdowns. All the data is available in the Leap toolkit, but I have not implemented the analyses.

### **6.3.5 Replication of the study and the Reflective Software Engineering class**

The sample size for the empirical study was small, just 16 students. Are the trends that I found in this study general? Replications of this study with other students and industrial developers would help us understand if there are generally more accurate estimation methods.

The case study gave me an opportunity to learn more about how the students used the Leap toolkit. It provided me with good insight into some of the issues the students had with learning software engineering concepts and incorporating them into their own development practices. The insights from the case study could lead to better teaching practices.

### **6.3.6 Investigations into FTR and personal data collection**

I believe that the defects that the reviewers find during review are more important for developer improvement than the defects that the developer finds during development. One area for further research is to find out if this is true. The Leap toolkit allows researchers to collect both the defects found during review and the defects that the developer makes. By using these defects and interviewing the developers, we could find out if the defects found by reviewers are actually more valuable than the defects the developer records.

Another area of interest is what types of defects should be collected. In the ICS 613 class, Dr. Johnson presented many different defect collection strategies. He told the students to experiment with using the different collection strategies. Watts Humphrey recommends that every defect should be recorded. He says, “Keep this form [defect recording log] on hand during compile and test. When you encounter a defect in compile, for example, enter a defect number at the next blank entry in the log.” We could investigate the benefits and costs of the different strategies.

## **6.4 Lessons for current software development improvement practice**

Some of the commonly used software development improvement practices include the PSP, the Capability Maturity Model (CMM), Formal Technical Review (FTR), and the Experience Factory. These approaches all collect data about the workproduct, and process. Most of them collect the data on paper forms, although there are many tools to help automate some of the data collection. This research has many implications for current software development improvement efforts. Some of them are:

- *Do not underestimate the overhead involved in measurement.* While empirical measurement has many benefits, it does introduce overhead to the user. Adding an additional measure to a developer's process may have a huge psychological overhead for the developer. As this research shows higher overhead leads to collection error.

The PSP is a good example of a method that has a high measurement overhead. Each time the developer finds a defect they must stop their work and record the defect in their Defect Recording Log.

- *Design improvement methods to allow multiple "entry points" into data collection.* The developer should be able to collect the data that will help them focus on what is important right now. The method should not try to record everything, because the additional overhead of data collection will reduce the quality of the data collected.

Again, the PSP is a good example of a developer improvement method that does not have multiple entry points for data collection. From the first project the developer must record all their size, time, and defect data. If they do not record all their data then the PSP's analyses are almost useless.

- *Address the issue of Measurement Dysfunction.* If the improvement effort does not address measurement dysfunction then the data collected may not reflect what is really happening and changes to the development process may cause additional harm. To reduce measurement dysfunction, keep the data private, or separate it from organizational data, or make the data anonymous or aggregate the data before it is shared. For example, one industrial user group promoted the integration of the Leap toolkit's defect collection mechanism with their organizational defect tracking system. I stopped this approach since it would lead to measurement dysfunction in the developer's defect data.
- *Focus improvement efforts on the individual developer.* They are the ones who are the key to high quality software development. High quality developers develop high quality software.

## 6.5 Implications for future software development improvement efforts

This research also has many implications for future software development improvement efforts. Some of them are:

- *Automated support is very important.* Future improvement efforts must include automated support to reduce the overhead for the software developer. Any improvement effort whose cost to the developer exceeds the benefit to the developer will require large amounts of management support to be adopted by the developers. The Leap toolkit demonstrates that automated support can reduce overhead.
- *Developers must learn from their experiences.* It does not make sense to collect data about the development process during the improvement effort just to collect the data. For example, the defect data collected in most reviews is rarely analyzed and disseminated back to the developers.
- *Incorporate data from different sources for the improvement effort.* For example, some very valuable sources of defect data are group reviews, software testing and customer reported defects. All of these sources can be used in the individual developer's defect analysis.
- *Do not focus on traditional productivity measures.* Using productivity measures for evaluation leads to measurement dysfunction. Focus instead on the developer's ability to complete the project on time and with good quality.
- *New improvement efforts must work in existing environment.* Show the developers how the new improvement works in their current environment. Training for new efforts should be done on existing projects. Developers should modify the way they work to the new process.
- *Paper is bad.* Recording data on paper is bad, because when the data is on paper it is very difficult to manipulate and analyze. The cost of data analysis for paper data is too high. Since the data cannot be easily analyzed, it just gets stored and ignored.



## **Appendix A**

# **Leap Evaluation Surveys**

## Leap User Survey #1

### Instructions

Please answer all the questions to the best of your ability. No not put your name on this survey. When you are finished filling out the survey place it in the envelope provided. Seal the envelope and sign your name across the seal. This will ensure that no one tampers with your survey. The envelopes will not be opened until after the grades for this class are turned in.

### Personal Programming Experience

Please answer the following questions about your programming experience. Give your best answer. Your answers do not have to be exact.

1. Roughly how many years of programming experience do you have? \_\_\_\_\_ (years)
2. Roughly how many years of *Java* programming experience do you have? \_\_\_\_\_ (years)
3. Roughly how many total thousands of lines of code (KLOC) have you written in any language?  
\_\_\_\_\_ (KLOC)
4. Roughly how many KLOC of *Java* have you written? \_\_\_\_\_ (KLOC)
5. Roughly how many different programming languages have you used? \_\_\_\_\_ (languages)
6. Roughly how big (in KLOC) is the biggest program (or your part of a program) that you have written?  
\_\_\_\_\_ (KLOC)
7. Roughly how big (in KLOC) is the biggest Java program (or your part of a program) that you have written? \_\_\_\_\_ (KLOC)
8. Roughly how many graduate level ICS classes have you taken? \_\_\_\_\_ (classes)
9. Roughly how many years of paid professional programming experience do you have? \_\_\_\_\_ (years)
10. Roughly how many different paid professional programming jobs have you had? \_\_\_\_\_ (jobs)

### Software Engineering Experience and Attitudes

Please circle the number that most closely matches your feelings about the following statements. If the statement does not apply to you circle NA.

	Strongly Disagree					Strongly Agree					
	1	2	3	4	5						
I am familiar with many different software development processes.											NA
I have used many different software development processes.											NA
I understand Object Oriented Programming.											NA
I can explain the benefits of Object Oriented Programming to other programmers.											NA
I know what the Personal Software Process is.											NA
I have used the Personal Software Process.											NA
I am aware of my own software development process.											NA
I am a good software developer.											NA
I want to be a better software developer.											NA
I can explain what it means to be a good software developer.											NA

Figure A.1. Leap survey #1 page 1

### Leap Usage – Time Collection

Please circle the answer that most closely matches your use of the features in Leap. Choose NA if the question does not apply to you.

Questions	Answers					
Which time entry tool do you use most often?	Io (single line entry tool)			Naia (time table)		
How often do you use Io (single line) to record your time?	Never (0%)	Less than half the time (1 – 39%)	About half the time (40 - 60%)	More than half the time (61 – 99%)	All the time (100%)	NA
How often do you use Naia (table) to record your time?	Never (0%)	Less than half the time (1 – 39%)	About half the time (40 - 60%)	More than half the time (61 – 99%)	All the time (100%)	NA
I use Naia to edit my time data.	Never (0%)	Less than half the time (1 – 39%)	About half the time (40 - 60%)	More than half the time (61 – 99%)	All the time (100%)	NA
I prefer to use Io for time recording.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I prefer to use Naia for time recording.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
My Leap data accurately reflects what really happened	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA

Please list the most negative aspect(s) of the Leap toolkit, in your opinion.

1.

2.

3.

Please list the most positive aspect(s) of the Leap toolkit, in your opinion.

1.

2.

3.

If you would like to provide any additional comments about the Leap toolkit, please note them on the rest of this page and the back of this page. Thank you for taking the time to fill out this survey. Please place it in the envelope, seal and sign the envelope.

Figure A.2. Leap survey #1 page 2

## Leap User Survey #2

### ***Leap Usability survey***

Please circle the number that most closely matches your feelings about the following statements. If the statement does not apply to you circle NA.

	Strongly Disagree					Strongly Agree	
	1	2	3	4	5		NA
Overall, I am satisfied with how easy it is to use Leap.							
It is simple to use Leap.							
I can effectively complete the Leap portions of my assignments.							
I can efficiently complete the Leap portions of my assignments							
It is easy to learn to use Leap.							
I believe I became productive quickly using Leap.							
Leap gives error messages that clearly tell me how to fix problems.							
Whenever I make a mistake using Leap, I recover easily and quickly.							
The information (such as on-screen messages, and other documentation) provided with Leap are clear.							
It is easy to find the information I need.							
The information provided with Leap is easy to understand.							
The information is effective in helping me complete the tasks and assignments.							
The organization of information on Leap screens is clear.							
The interface of Leap is pleasant.							
I like using the interface of Leap.							
Leap has all the functions and capabilities I expect it to have.							
Overall, I am satisfied with Leap.							

Figure A.3. Leap survey #2 page 1

### Leap Usage – Time Collection

Please circle the answer that most closely matches your use of the features in Leap. Choose NA if the question does not apply to you.

Questions	Answers					
Which time entry tool do you use most often?	Io (single line entry tool)			Naia (time table)		
How often do you use Io (single line) to record your time?	Never (0%)	Less than half the time (1 – 39%)	About half the time (40 - 60%)	More than half the time (61 – 99%)	All the time (100%)	NA
How often do you use Naia (table) to record your time?	Never (0%)	Less than half the time (1 – 39%)	About half the time (40 - 60%)	More than half the time (61 – 99%)	All the time (100%)	NA
I use Naia to edit my time data.	Never (0%)	Less than half the time (1 – 39%)	About half the time (40 - 60%)	More than half the time (61 – 99%)	All the time (100%)	NA
I prefer to use Io for time recording.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I prefer to use Naia for time recording.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
My time data gives me valuable insights into my strengths as a programmer	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
My Leap data accurately reflects what really happened	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
My time data gives me valuable insights into my weaknesses as a programmer	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA

### Leap Usage – Time Estimation

Please circle the answer that most closely matches your use of the features in Leap. Choose NA if the question does not apply to you.

Questions	Answers					
I am aware of my time estimate while I do my assignment.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I feel comfortable with my time estimates.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I feel comfortable with my ability to estimate project size.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I feel pressure to make my actual effort match my estimated effort.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I feel pressure to make my actual project size match my estimated project size.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I feel pressure to turn in data where my actual size and time data matches my estimates.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Time pressure reduces the quality of the data I collect in Leap.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA

Figure A.4. Leap survey #2 page 2

Please list the most negative aspect(s) of the Leap toolkit, in your opinion.

1.

2.

3.

Please list the most positive aspect(s) of the Leap toolkit, in your opinion.

1.

2.

3.

What have you learned about your own software development process?

If you would like to provide any additional comments about the Leap toolkit, please note them on the rest of this page and the back of this page. Thank you for taking the time to fill out this survey.

Figure A.5. Leap survey #2 page 3

## Leap User Survey #3

### Leap Usage – Time Collection

Please circle the answer that most closely matches your use of the features in Leap. Choose NA if the question does not apply to you.

Questions	Answers					
Which time entry tool do you use most often?	Io (single line entry tool)			Naia (time table)		
I prefer to use Io for time recording.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I prefer to use Naia for time recording.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I use Naia to edit my time data.	Never (0%)	Less than half the time (1 – 39%)	About half the time (40 - 60%)	More than half the time (61 – 99%)	All the time (100%)	NA
My Leap data accurately reflects what really happened	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
My time data gives me valuable insights into my strengths as a programmer.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
My time data gives me valuable insights into my weaknesses as a programmer	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA

### Leap Usage – Time Estimation

Please circle the answer that most closely matches your use of the features in Leap. Choose NA if the question does not apply to you.

Questions	Answers					
I am aware of my time estimate while I do my assignment.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I feel comfortable with my time estimates.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I feel comfortable with my ability to estimate project size.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I feel pressure to make my actual effort match my estimated effort.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I feel pressure to make my actual project size match my estimated project size.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I feel pressure to turn in data where my actual size and time data matches my estimates.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA

Figure A.6. Leap survey #3 page 1

My time estimation skills are improving.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
My size estimation skills are improving.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA

### ***Leap Usage – Defect Collection***

Please circle the answer that most closely matches your feelings. Choose NA if the question does not apply to you.

Questions	Answers					
Being aware of my defects helps me avoid making them in the future.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
My defect data gives me valuable insights into my strengths as a programmer.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
My defect data gives me valuable insights into my weaknesses as a programmer	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Collecting my defect data is a waste of my time.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
My Leap data accurately reflects what really happened	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Time pressure reduces the quality of the data I collect in Leap.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA

What have you learned about your project estimation abilities?

If you would like to provide any additional comments about the Leap toolkit, please note them on the back of this page. Thank you for taking the time to fill out this survey.

Figure A.7. Leap survey #3 page 2



## Leap User Survey #4

### *Leap Usability survey*

Please circle the number that most closely matches your feelings about the following statements. If the statement does not apply to you circle NA.

	Strongly Disagree				Strongly Agree				
	1	2	3	4	5				NA
Overall, I am satisfied with how easy it is to use Leap.									NA
It is simple to use Leap.									NA
I can effectively complete the Leap portions of my assignments.									NA
I can efficiently complete the Leap portions of my assignments									NA
It is easy to learn to use Leap.									NA
I believe I became productive quickly using Leap.									NA
Leap gives error messages that clearly tell me how to fix problems.									NA
Whenever I make a mistake using Leap, I recover easily and quickly.									NA
The information (such as on-screen messages, and other documentation) provided with Leap are clear.									NA
It is easy to find the information I need.									NA
The information provided with Leap is easy to understand.									NA
The information is effective in helping me complete the tasks and assignments.									NA
The organization of information on Leap screens is clear.									NA
The interface of Leap is pleasant.									NA
I like using the interface of Leap.									NA
Leap has all the functions and capabilities I expect it to have.									NA
Overall, I am satisfied with Leap.									NA

Figure A.8. Leap survey #4 page 1

### Perceptions of Leap

Please circle the answer that most closely matches your feelings. Choose NA if the question does not apply to you.

Questions	Answers					
Using Leap enables me to develop software more quickly.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Using Leap improves the quality of my software.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Using Leap makes it easier for me to develop software.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Using Leap improves my software development performance.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Overall, I find using Leap to be advantageous in my software development.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Using Leap enhances my effectiveness in software development.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Using Leap gives me greater control over my work.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Using Leap increases my productivity.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Using Leap is compatible with all aspects of my software development process.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I think using Leap fits well with the way I develop software.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Using Leap fits into my work style.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I believe that Leap is cumbersome to use.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
My using Leap requires a lot of mental effort.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Using Leap is often frustrating.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I believe that it is easy to get Leap to do what I want it to do.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Overall, I believe that Leap is easy to use.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I would have no difficulty telling others about the results of using Leap.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I believe I could communicate to others the consequences of using Leap.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA

Figure A.9. Leap survey #4 page 2

The results of using Leap are apparent to me.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I would have difficulty explaining why using Leap may or may not be beneficial.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA

### ***Lessons Learned***

Please circle the answer that most closely matches your feelings. Choose NA if the question does not apply to you.

Questions	Answers					
Leap has shown me valuable insights into my software development process.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
My programming skills have improved since the beginning of this class.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I will continue to use Leap as a part of my software development.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Collecting data about my software development process has been a waste of my time.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
My Leap data accurately reflects what really happened	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
Using Leap has made me a better software developer.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA
I can describe my own software development process.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	NA

Please describe the most valuable insights you have learned about your software development process.

Thank you for taking the time to fill out this survey.

Figure A.10. Leap survey #4 page 3

# Appendix B

## Survey Data

### B.1 Survey # 1

Table B.1. Survey # 1 Results: Personal Programming

Personal Programming Experience										
ID	1	2	3	4	5	6	7	8	9	10
1	4	0.5	10	2	2	3.00	1.00	3	2	1
2	7	2	NA	NA	8	1.50	1.50	6	0	0
3	10	0.2	5	1.5	7	5.00	1.50	2	6	2
4	2.5	2	10	2	3	2.00	1.00	10	2	2
5										
6	3	2	NA	NA	4	1.50	1.50	1	0	0
7	3	1	10	1.5	4	5.00	2.00	7	0.5	1
8	2	0.5	15	3	3	2.00	1.50	2	1	2
9	6	1.5	20	3	6	3.00	1.50	8	1	1
10	5	2	10	3	5	2.00	2.00	2	5	3
11	20	0.5	NA	NA	5	NA	NA	2	8	2
12	2	2	3	2	3	0.70	0.70	2	0	0
13	8	1			5			9	3	1
14	5	1	NA	2	7	1.00	1.00	6	0.5	1
15	3	3	100	10	5	10.00	2.50	7		4
16										

Table B.2. Survey # 1 Results: SE Experience and Attitudes

<b>SE Experience and Attitudes</b>										
<b>ID</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
1	2	2	4	4	3	1	1	3	5	4
2	2	2	2	2	1	2	2	3	5	3
3	2	2	2	2	2	1	4	3	5	4
4	3	3	5	5	3	4	4	5	5	5
5										
6	3	3	3	4	5	5	3	3	5	5
7	3	2	3	4	3	1	5	4	5	4
8	2	2	5	5	4	2	3	4	5	5
9	3	3	2	4	5	5	5	3	5	5
10	4	4	4	4	4		4	4	5	4
11	2	2	5	4	3	5	3	5	5	4
12	3	2	4	4	2	2	2	3	5	3
13	2	2	4	5	2	1	4	4	5	4
14	4	3	5	5	3	3	5	4	5	5
15	3	3	4	4	3	3	3	3	4	3
16										

Table B.3. Survey # 1 Results: Time Collection

<b>Time Collection</b>							
<b>ID</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>
1	I	5	2	2	5	2	4
2	I	5	2	2	5	3	4
3	I	4	2	3	5	2	4
4	I	4	2	2	5	3	5
5							
6	I	4	2	2	4	2	4
7	N	2	4	4	2	4	4
8		4	2	1	5	1	4
9	I	3	3	2	4	3	3
10	I	4	2	2	5	2	5
11	I	4	2	1	5	1	4
12	IN	4	2	4	4	3	4
13	I	2	2	1	4	2	4
14	N	2	4	5	3	5	5
15	IN	3	3	2	3	4	4
16							

Table B.4. Survey # 2 Results: Leap Usability

Leap Usability																	
ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	4	3	5	5	4	2	3	4	4	3	4	4	5	5	4	3	4
2	3	3	3	3	3	2	2	3	3	3	3	3	3	3	3	3	3
3	3	4	4	3	4	4	3	3	2	4	4	4	3	3	3	4	4
4	3	5	5		1	3	3	4	4	4	3	4	3	2	2	3	4
5	2	2	2	2	2	2	2	2	2	2	3	2	3	3	2	4	2
6	2	2	2	2	2	4	2	2	3	4	5	4	5	5	5	4	2
7																	
8	3	3	4		3	4	2	2	3		3	3	4	3	3	5	4
9	3	3	4	4	3	4	2	2	3	2	3	3	4	4	4	4	4
10	4	3	2	2	3	3	3	2	4	4	4	4	4	4	4	4	4
11	4	3	4	3	3	3	2	2	3	2	3	3	5	3	3	5	4
12	4	4		3	4	3	3	3	3	3	4	4	5	5	5	5	4
13	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
14	3	3	2	2	3	3	2	3	3	3	3	3	3	3	3	3	3
15	2	2	1	4		3	3	1	3	3	3	3	2	2	2	3	2
16	2	3	3	2	3	2	2	2	2	2	2	2	3	2	2		2

Table B.5. Survey # 2 Results: Time Collection

Time Collection									
ID	1	2	3	4	5	6	7	8	9
1	I	4	2	2	5	1	4	4	4
2	I	3	2	2	4	3	2	3	2
3	I	4	2	2	5	2	5	4	4
4	I	4	1	2	5	1	5	3	4
5	N	1	4	4	1	4	3	3	2
6	I	4	2	3	5	1	4	4	4
7									
8	I	4	2	2	4	2	4	4	5
9	N	4	5	5	4	4	4	4	4
10		4	2	2	4	3	4	3	4
11	I	4	2	2	5	1	4	4	4
12		4	4	4	5	4	4	4	5
13	I	4	2	2	4	3	4	3	3
14	N	2	4	5	2	5	4	4	4
15		3	3	3	3	3	4	4	3
16	N	3	4		3	3	3	2	2

Table B.6. Survey # 2 Results: Time Estimation

Time Estimation							
ID	1	2	3	4	5	6	7
1	5	2	4	5	2	1	2
2	3	3	3	5	5	5	5
3	4	4	2	4	2	2	2
4	4	4	4	3	3	2	2
5	1	2	1	2	3	4	2
6	3	4	2	3	3	2	2
7							
8	4	4	3	3	3	2	2
9	4	3	4	4	4	4	3
10	4	3	4	4	3	3	3
11	5	3	4	3	2	2	4
12	4	4	4	3	3	3	4
13	4	3	4	4	2	3	3
14	4	4	4	4	4	4	3
15	3	3	2	3	3	3	3
16	4	3	3	4	3	3	3

Table B.7. Survey # 3 Results: Time Collection

Time Collection							
ID	1	2	3	4	5	6	7
1	I	5	2	2	4	4	4
2	N		3	3	3	2	2
3	I	5	2	2	5	4	4
4	I	4	2	2	4	3	3
5							
6	I	5	3	2	4	4	4
7	N	3	3	3	4	4	4
8	I	4	2	2	5	4	4
9	I	4	4	4	4	3	4
10		3	4	4	4	4	4
11	I	5	1	2	4	3	4
12	I	5	4	4	4	4	4
13	I	5	3	2	4	4	4
14	N	3	5	4	4	4	4
15	IN	3	4	3	4	4	3
16	N	3	3	4	2	3	3

Table B.8. Survey # 3 Results: Time Estimation

Time Estimation								
ID	1	2	3	4	5	6	7	8
1	5	4	3	4	1	2	4	4
2	3	3	3	5	5	5	3	3
3	5	4		2	2	2	4	4
4	4	4	3	4	3	3	4	4
5								
6	4	4	3	2	3	2	4	3
7	3	3	3	2	2	2	4	3
8	4	4	4	2	2	1	4	4
9	3	3	4	5	4	3	4	4
10	4	4	4	3	3	3	4	4
11	3	3	3	2	2	1	3	3
12	4	5	4	3	3	3	5	5
13	5	3	4	4	3	2	4	4
14	4	4	4	3	3	3	4	4
15	4	3	3	2	2	2	3	3
16	4	3	3	4	3	2	3	3

Table B.9. Survey # 3 Results: Defect Collection

Defect Collection						
ID	1	2	3	4	5	6
1	4	4	4	1	4	4
2	3	3	3	3	3	4
3	5	4	4	1	5	4
4	3	3	3	4	4	2
5						
6	4	4	4	2	4	2
7	4	4	4	4	4	2
8	4	4	5	1	5	2
9	4	4	4	3	4	3
10	4	4	4	1	4	4
11	4	2	4	2	4	4
12	3	4	4	2	4	4
13	4	4	3	3	4	3
14	4	4	4	2	3	2
15	4	4	4	3	4	2
16	3	4	3	3	3	2



## B.2 Survey #2

## B.3 Survey #3

## B.4 Survey # 4

Table B.10. Survey # 4 Results: Leap Usability

Leap Usability																	
ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	4	2	4	4	2	2	4	4	4	2	4	4	4	4	4	4	4
2	3	3	3	3	3	2	2	3	3	3	4	2	3	4	3	3	2
3	4	3	4	4	4	4	3	3	3	3	4	5	4	4	4	4	5
4	3	3	4		4	4	4	2	4	3	3	4	3	3	3	4	4
5	3	3	4	4	3	3	2	3	3	3	3	3	4	4	4	4	4
6	2	2	3	3	3	2	1	2	2	5	5	4	5	4	4	4	3
7	4	3	4	4	5	3	3	3	4	4	4	3	4	3	3	3	4
8	5	4	5		4	5	4	4	4	4	5	5	5	5	5	5	4
9	4	4	5	5	4	4	3	3	3	3	4	4	5	4	4	5	4
10	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	4	4
11	3	3	4	3	3	3	2	3	3	3	2	3	4	4	3	3	3
12	5	5	5	5	5	5	4	4	5	5	5	5	5	5	5	4	5
13	4	3	3		4	4	4	4	3	4	4	4	4	4	4	3	4
14	4	4	4	4	4	4	3	3	3	4	4	4	4	4	4	4	4
15	4	4	4	4	4	3	3	2	4	4	4	3	5	5	5	4	4
16																	

Table B.11. Survey # 4 Results: Perceptions of Leap

Perceptions of Leap																				
ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	2	3	4	2	4	4	4	2	4	4	3	2	2	2	4	2	4	4	2	2
2	2	2	1	2	2	2	1	1	3	2	1	4	2	4	3	3	4	3	3	3
3	5	5	4	4	5	5	5	4	4	5		2	2	2	4	4	4	4	5	2
4	4	3	3	4	4	4	4	4	3	4	4	3	2	3	4	4	5	5		1
5	3					2	2	2	4	3	2	3	4	3	4	4	4	2	2	3
6	2	4	4	4	4	4	5	4	4	3	3	4	4	4	4	3	4	4	4	2
7	3	4	3	4	4	4	3	3	3	4	3	4	4	3	3	4	4	4		4
8	4	4	4	5	5	5	5	4	4	5	4	2	1	1	4	5	4	4	5	2
9	4	4	3	4	4	5	5	4	4	5	4	2	3	3	4	3	5	5	5	2
10	4	4	4	4	4	4	4	4	4	4	4	2	2	2	3	4	4	4	4	2
11	2	4	2	4	4	4	4	3	2	2	2	4	3	3	2	2	4	4	4	2
12	4	5	4	5	5	5	5	5	5	5	4	4	3	3	4	4	5	5	5	2
13	4	3	4	4	4	4	3	4	3	4	3	3	4	4	3	3	4	4	4	3
14	4	3	3	4	4	4	4	4	4	4	3	3	3	2	4	4	4	4	4	3
15	3	4	3	4	4	4	4	3	4	4	3	3	3	3	3	4	4	3	4	3
16																				

Table B.12. Survey # 4 Results: Lessons Learned

Lessons Learned							
ID	1	2	3	4	5	6	7
1	4	3	4	2	4	4	4
2	2	3	1	3	3	2	3
3	5	4	4	1	4	4	5
4	5	3	5	1	5	5	5
5	3	4	3	2	3	4	
6	5	5	5	2	4	5	4
7	4	4	3	4	4	4	4
8	5	5	5	1	4	5	4
9	5	5	4	2	3	4	4
10	4	4	4	2	4	4	4
11	4	5	4	2	4	4	4
12	5	5	5	1	5	5	5
13	4	4	4	2	4	4	4
14	4	4	3	3	4	4	4
15	4	4	3	3	4	4	4
16							

## Appendix C

# Student Interviews

I was only able to interview 11 out of the 16 students in the class. The following are edited transcripts of the interviews. I have organized the the answers by question. In the answers by the students, my additional questions are *emphasized*.

### **C.1 How do you feel about collecting size, time, and defect data about your software development process?**

**Student #1:** Well, I think it is very necessary. It help you to gain some insight into your abilities to develop software.

**Student #4:** I think it is very good. Actually, besides this one I'm doing it on ... other programming projects.

**Student #5:** Well, I think it is very good. You should collect size time defect data.

**Student #6:** I think that collecting the size data and you know time data is valuable but, I think the defect data is really not too useful. Maybe in something, somehow you use it.

*So, collecting defect data is not too useful?*

yea

**Student #8:** Well, I get used to it and I know how to use it. I feel that I enjoy because it is just something that is very good. And, I think, I never thought about it but, I think that anyone who might have some experience, and after a while without this kind of system, kind of tools, you use,

you know, that collect data and, I don't know collect exactly or precisely. But, before I used leap and reflective programming, I say oh thats going to take a long time. But how long? This project is going to be big but, how big and without that information. So, that is why I think this is a good thing, that I will use in future programming. The only thing that is a little bit, you know, bad, that, in a new system, in the beginning you have no information. So, only after a certain amount of time or, you know, you have done a few small project, you are really able to do planning and estimation but, other wise just in the beginning just collect data. But, if you work in a certain area, at this point, you have got to do that in the beginning. After all, you know, what are you capabilities and how you do things.

**Student #9:** I think size data is very helpful. But, I have a problem just to, well, I input the size and I like want to a connect it to the project. But, I think it must be the file location must be the same as the... I think it is a little not easy to know because, last time I spent a lot of time just to get \*.java. So, the times I think is very good.

*Ok, how did you collect defects?*

Yea, defects is, defects is also good.

**Student #11:** Well, I'm of two minds at least you know. Its kind of a pain but, trying to do the last project where we're non java stuff setting it all up its really been kind of of .. I've been having trouble getting motivated and you know. Sometimes even the stuff that is all set up...like why do I have to do this. I don't want to do this leave me alone do this automatically. Just just look at what I'm doing and see what I'm doing. Don't make me tell you you know. It is important so sometimes I'm more motivated and you know. I certainly want to be able to make good estimates and I hope to be heading in that direction. So, there you have it.

**Student #12:** I think that it is very useful and helpful. This is the first time I use the Leap. Even, I just begin the Leap and now I can use it very well.

**Student #13:** I think it is very useful, because before I didn't know my, how I mean, I can, you know. I don't have that kind process, not just the one process, I had some process it wasn't very clear.

**Student #14:** You mean before I develop the software or after I develop. Because, I do not use Io to record my time. I always use the Naia to record the time but, normally I would record my time after I finish. And then I would record the time according to my remembrance.

I think it costs a lot of work because normally I would have to spend around 40 minutes to finish all of that. Maybe that is because my computer is not that fast and I have to wait.

*Collecting the data was sort of hard because it added that time at the end?*

Yea, because I had to add the begging time and the end time of each time. But, I think for the defect it is OK.

*Oh, did you do that while you were programming? Record the defects?*

No, I just record the defects on a note then at the end of the program then record them.

**Student #15:** I think it is very good for me. To have good experience, To collect the size and the time and record the size and time and then we can get on my own data to make the prediction. Yea, which is, I think. Thats very important because you can have the the raw estimate for the new project.

## **C.2 Was collecting the Leap data easy, neutral, or hard?**

**Student #1:** I think it is neutral. It is not very easy. You have to pay some attention. It is not very hard either.

**Student #4:** I think it is a quite easy for me. Because, I had this experience last semester. Yea, I think once, you know, get used to it. It should be fine. But, sometimes I, you know, last time I create a new Leap project for my visual basic programming and I had some problems like the the cell should be editable but I could not type in data.

**Student #5:** Well, at first I think it is hard and long time. Then easier.

**Student #6:** Collecting the time data and the size data was easy but, the defect data. So, when you are programming sometime you do not want to record all the defects.

*So, stopping to record the defect was too bother, too hard.*

Yea, maybe it is because when you are debugging you do not want to think about other things.

*So, you want to keep working instead of having to stop.*

Yea.

**Student #8:** Very easy.

**Student #9:** I think it was neutral because Leap is very slow.

**Student #11:** Well, certainly easier than if I was doing it the Watts Humphrey's way. But, still not easy enough.

**Student #12:** I think very easy.

**Student #13:** Well, it is not supposed to be that hard, but sometimes the, it works that you know very... Sometimes, some paths, when you set the path of the project. If you, just sometimes the... I don't know why it doesn't work but I have to struggle for a while to make it work but it is supposed to be easy.

**Student #14:** I think it costs a lot of work because normally I would have to spend around 40 minutes to finish all of that. Maybe that is because my computer is not that fast and I have to wait.

*Collecting the data was sort of hard because it added that time at the end?*

Yea, because I had to add the begging time and the end time of each time, but I think for the defect it is OK.

**Student #15:** I think it is between the neutral and the easy. It is not really difficult yea.

### **C.3 Did collecting all this data change the way you program?**

**Student #1:** I think it had some affect because I felt some pressure. I made the estimate. With time collection, I will try to reach the goal. But, if I don't worry about time recording, I maybe relax more and then maybe more longer code, change the formating of the code. But, I know I have a goal there. I just try to finish job. Get the job done. Try to get the program running. That's what I do but if don't do it, maybe I spend more time on restructuring the code, refactoring the code something like that. But, I think it changes the somehow it changes my behavior as I record.

**Student #4:** Change the way I program. Yea, I think so. Yea, I could be, I could, I'm focusing my attention more. Attention on the project. That is what I'm thinking. Makes me think I'm managing myself.

**Student #5:** Kind of, a little bit, not too much.

*In what sort of ways did you change because you were collecting this data?*

Well, simply because I collected defects so I know what kinds of defects I make and what kinds of mistakes I will make.

**Student #6:** yea

*How did it change?*

I think when the time data. We record the time data when change the phase. So, I think that sometime I spend time, even more sometimes. I bother to record but, I'm thinking about the time.

*So, you are thinking about how you were doing the programming because you had to record a phase?*

Yea, thats why.

**Student #8:** No, not at all.

*not at all, ok.*

Yea, because one thing is due to this professor, Dr Johnson, said he just feels that if you finish this project you know more than how much time. I might feel pressure either by, you know. Push my self harder or, you know, just do some fairly quick data, or I fake some data. But, he said in the beginning clearly he says you know doesn't matter. That not an issue. So, I don't see any change. And, I, it is, just, a record of yourself. You don't competing, you know. You think you try to beat you. That is not the point.

**Student #9:** Yea. I think so.

*how?*

Especially, I think the time data you pay attention to the project. So the actually you do quickly.

**Student #11:** I don't think so but, maybe that is bad. That one of the things I want to talk about later.

**Student #12:** I think one thing it changes can help you improve.

*By collecting the data you improved?*

Yea, you improve. Help you improve your programming. Just like some size or time. You know before you have to feel how big the program is. But, now we use Leap. We can know exact how many lines, how many methods, how many classes. But before, I know the classes, I know the method but, lines of code ugh hard to know because we didn't have Leap.

**Student #13:** I think so. A little bit yea.

*How did that, what how did it change the way you program?*

Like, I actually predict my phases. Do it in different phases. But, before I just a do it a together. Maybe I. Maybe, I do design but, I don't do it really realize it. I did it more.

**Student #14:**

**Student #15:** I don't think so.

## **C.4 Do you think your Leap data accurately reflects what really happened?**

**Student #1:** I think so, maybe 95%. Because sometimes I just stop it. Go do something else and when I come back I have forgot. Then I try to edit follow up and focus on the data again. I can't think 100%.

**Student #4:** I'd say sometimes not really because, you know, sometimes I'm thinking about things but I didn't count the time.

*Ok, so like doing some sorts of design or thinking about the problem you're not recording the time?*

The design phase, I think though take a huge part of the time because you know. Once we got the project we were designing our GUI but we didn't count it in the Leap.

**Student #5:** No, I think the leap data is not very accurate it is not very accurate for the estimation of the size because the lines per method is not the same for all the methods.

*How about the actuals that were collected like when you recorded your time. Do you think that was accurate? I mean when you recorded time, thats really the time you were spending?*



Roughly, because sometimes if I run out I didn't record it.

*And the defects? Do you think that is what really happened?*

Yea, although it is not all the defects I didn't record syntax.

*And for the defect. You didn't collect all of them was it because it was too hard to enter the data or is it takes too much time?*

Too much time.

**Student #6:** Most of the time, maybe sometimes the leap data is not recorded.

*Time data or what kind of data is not recorded?*

Usually, most of the defect data was not recorded and a some time data. Sometimes you work for a very short time you don't want to bother to record it.

**Student #8:** Most of the time the only thing that I find is the planning. Because when I did design and then back in the beginning I had a little bit worried about. Since, if I don't have anything about this project, how do I do design? how do I estimate how much time I need to design? That's hard. That's something that I had a question on. I think you can choosing class and raise that question.

**Student #9:** Most times but, sometimes about the size. Maybe I just change the class name or something and it says big change.

**Student #11:** Pretty accurately. There were a few, you know, oops I forgot to record that. How much time was it? Well lets say 30 minutes. But, mostly I remembered to to it and not too many crashes that destroyed data.

**Student #12:** Yea. I think.

**Student #13:** Basically.

*What would make the data not match what really happened?*

Probably like I forgot something. Sometimes when I'm interrupted I forgot to click it.

*So, for the defects sometimes maybe you didn't record every defect you made?*

Yea, and I maybe I didn't record the time. I just estimate how long, there is a reason for that.

*The reason is?*

That, my version of Leap. When I use the export function, there is no Io recording the time that button “save” something disappeared. But, I didn’t download the new version because it takes a long time. So, I just used the bad version so I couldn’t actually accurately record the time.

*So you sometimes you couldn’t use Io to record your time so then...*

I use the times but for the times sometimes I forgot.

*And for the defect data you didn’t record everything is there a reason or is it too hard to record the data or?*

It is just too, you have to interrupt. It is too much trouble.

*It is easier to just fix the problem and go on?*

Yea, that is my way. I really don’t like to record defect. I really don’t feel that it is very helpful because each time I have a different.

*different kind?*

Yea, even though I know that kind of problem like null pointer. That is always a problem, but for different times null pointer is caused by different reason so you just cannot.

*It is hard to just say null pointer and know how to fix?*

Yea, it is hard to say it and so how useful to record it, you know it. You know the problem but you don’t know how to improve it.

**Student #14:** Yes,

*So you think your recording was what really happened?*

Not exactly, but it is accurate.

*Close?*

Close.

**Student #15:** 80% at least. I think.

*So what happened in that 20 percent that wasn’t? The data was different than what really happened?*

That, the reason I gave that because sometimes I using the editor to edit the time. Yea, then during the daytime when I’m working because I have a job. So that time I’m not collecting.

*So you were working without using Leap and you went back and recorded as you remembered as you did it?*

Yea.

## **C.5 How about the size data from LOCC, do you think that it was accurate?**

**Student #1:** I don't know actually I never questioned the results. I never checked. if it told me how many line per method you couldn't manually check I just accepted it.

*So it felt right?*

Yes.

*it felt like the right numbers?*

For just 1 class no problem, for 20 method, for 1 class it is right, for 20 method is close, for number of lines I have no idea.

**Student #4:** I assume. I think it is accurate, but I you know I don't know how exactly. For the new files, I think it is perfect but for the old ones I modified something I don't know how the arguments made is too complex.

*But it feels the amount of work you did and the numbers it comes out is seems ok?*

Yea, it seems ok.

**Student #5:** Accurate.

**Student #6:** Yea, they were accurate. Yea, sometimes I make a very small change add two or three lines of code the Leap actual data recorded was two and usually it was accurate.

**Student #8:** Well, I trusted it. I never, I didn't try another way or count it by myself but i think that should be.

*So it felt like when you did some work between projects three and four or what ever, the amount of work felt about the size that LOCC you said you did?*

Oh yea, of course. Yea, but the only thing that I, that seemed to me is. From now on, I only use class and package, you know, size any more. That is just, you know, something you have no idea at all how big for class how many of them are there in the program. Lines of code or methods are good.

**Student #9:** *Ok, so the size data was not quite accurate?*

Yep.

**Student #11:** At least in one case it was way off. Cause I big cut and paste thing and it was saying oh you've done massive changes. I said, "haha, I'm so brilliant." Not really but, still it helps you got to have something. I'm not going to count those lines myself good God.

**Student #12:** Yea.

**Student #13:** It is very useful. Size and time is very useful. I don't know. I just used it but I don't know whether it is accurate or not.

**Student #14:** It feels Ok. I think it is Ok.

*So it seemed that the numbers it was giving you*

*yea*

*sort of feel right?*

Yes, but you know sometimes I, maybe just modify something but I do not make much change to that class, but it also count that I modified a new class.

**Student #15:** Yea, that is because really depends on sort of how you run the software right. I found that if you run it twice you have a problem yea but better than LOCC.

## **C.6 How do you feel about making an estimate of how long a project will take?**

**Student #1:** I think it is very hard. In the beginning you know for first few projects, the command line parser. When Philip told us the functionality, actually I can design really clearly how many methods. But later on, when it comes to the user interface actually I have no idea how many methods because it involves inner classes. How do you say the inner class is a method or is a class. Something you just. The way I did it, I take a look at other programs user interface programs and essentially the complexity is similar, so I count how many methods I used and how many lines of code. So I get a rough idea from other programs to compare but I didn't follow your process tracking your programs I didn't use your method. It was easier.

**Student #4:** It is necessary.

*Its necessary. Does that put a lot of pressure on you?*

No, I think you know what I think it depends on different people. You know, some people just like to programming and wait for others to define the project. Those kind of people if you ask them to do the time estimate or something they might feel pressure and don't like it. But for me, you know, I kind of like to manage something and a it is fine. It is kind of fine job for me.

**Student #5:** It just about an idea ... just how.

**Student #6:** I think that it is very hard to make an estimate because a usually my design wont go to such detail. Maybe I know there is some method but I don't know how long the method become. So just cant determine. It depends on the situation

*So that so to make a good estimate you should have a more detailed knowledge of how big the thing will be?*

Yea.

*And you design does not quite get to that level of detail?*

Yea, yea.

**Student #8:** I am comfortable. Every time I was, you know, I never did. I never did... I just want to say that at this time, I know spent a phase to read the code in the beginning. I spent a lot of time or many time because one thing is I'm not very good a Java and a lot of thing. And another thing is that sometimes, I really don't have much experience with GUI, you know. Thats in the in the past I learned a little about the classes.

**Student #9:** I always under estimate but it is still too difficult.

*So it is not too hard to make the estimate?*

Yea.

**Student #11:** It terrifies me. Well, you know, using it for Philip's class doesn't terrify me. But, you know, when I used to do this for a living and when I'm going to do it for a living again, you know, It just, it make me very ... Anxiety is involved. Almost if you get it wrong thats really bad. Thats really bad.

**Student #12:** Yea, I think that helps.

**Student #13:** I think that's useful to give me a concept of how long it is going to take me on that project plan.

**Student #14:** So far, I feel it is Ok. Because, you know, if I can accurate estimate my time, I can arrange my schedule so it is a good idea.

**Student #15:** No, I think it is good to estimate my next project size and time because everyone wants a plan that what time they want to spend for the next week for the next few days. I think it is very good.

## **C.7 Were you aware of your estimate while you were programming?**

**Student #1:** *See answer to "Did collecting all this data change the way you programmed?"*

**Student #4:** Yes.

*Did you change did you consciously try to a work faster work slower to meet that estimate?*

If, you know, I think I'm working slowly I'll try to make faster to make it but sometimes if I estimated too much I wont.

*You're not going to slow down to make it?*

No, no.

**Student #5:** I was thinking about it. I know I couldn't just program right.

**Student #6:** No.

**Student #8:** No, no.

**Student #9:** Yea.

*So you were thinking about I say...*

I always try to meet it.

**Student #11:** Not really. Not during the class, no. I was not paying much attention to it.

**Student #12:** Before I don't think. I don't think too much, but now we use Leap, we get the estimate. So I get that data we should think about it.

**Student #13:** Sometime it is pressure like in previous projects. I do realize it and I do try to meet that estimation but I think now I just a don't care.

**Student #14:** No

*No, you didn't weren't thinking like I Ok...*

I mean, I didn't think about the times but I think about the class and methods.

*You were thinking about the design you came up with?*

Yea, but I don't think about time.

**Student #15:** No.

## **C.8 How much pressure did you feel to make your actual time match your estimate?**

**Student #1:** *See above answer.*

**Student #4:** Not too much pressure. But, I, you know, we are students. We try to get in the real world and make some money on this, I think I will have some pressure if I made a bad estimate. I will try to work hard.

**Student #5:** Just working.

**Student #6:** Actually, usually I didn't feel pressure. I just recording my time schedule. If I have more time, I spend more time to make the program right. So just depending on my time schedule.

**Student #8:**

**Student #9:** I have no pressure.

**Student #11:** Sure there was Philip's dead lines which were pretty rubbery anyway.

**Student #12:** I just did it.

**Student #13:** I think, I didn't want to. I don't like that type of pressure to influence my programming. I want to do it better but not to.

*You don't want to just meet a time?*

No, I want to do it. I like my programs to be, the quality is the most important thing.

**Student #14:**

**Student #15:** I don't have that pressure. I just focus on what I'm doing.

## **C.9 In general, how accurate were your time estimates?**

**Student #1:** Most of the time is 30%. It is 30%.

**Student #4:** Its all right. Yea, we just analyzed the data. It is all right.

**Student #5:** In general, I think the error was maybe 20%.

**Student #6:** Not not very close, because my time data is... When I analyze my time data I found that the time data is usually very match the size data. But my estimate usually not very accurate only have about 20 or 30% error

*That is very good.*

But not more that 50-70 percent.

**Student #8:** For, I think that that project seven was the best one. Project eight not bad but is somehow it a not good. Sometimes can be like 30 or 50%. But my time estimates is some how I related is really correlated to my size estimate. And the problem is I do size estimation first. So, if that gets wrong, I have no way to avoid it. So my problem is size estimation and for size estimation one time it might be very close. But because of the way that I do it, the special case, here we copy some of the old files just by cut to the new files. Actually, we didn't do anything. We don't from there modify them, but when we use LOCC it counts them as totally new. So the size is the real size measurement is not given that really the file but that will not help me you know for the class when you do a project.



**Student #9:** Just gave 30%.

**Student #11:** Time, it really varied. Frequently not very good a couple of times I got close.

**Student #12:** You know, the first time you estimate something may be very different you. But, after a couple of times you should think about some part if you implement. Implement some place you are familiar so you can adjustment make the estimate. Yea, time a little longer but for certain parts. If the design estimate lets you know how much time you take, that not changing too much. Another thing I think, if we implement, I think we have to collect the data direct hours. So someplace some part we are not familiar we have to learn. I think you cannot collect that time in your program. If you put that time inside that, that time is different. If you just put the direct hours, I think you know sometimes you know Java is object oriented program. Sometimes, you know, we can just reuse some other persons class working for us. If so, even your code, you think will be the time you take not much.

**Student #13:** Sometime it was very good. Sometimes, but sometimes it depends.

**Student #14:**

**Student #15:** I think they were within 20% of of the variation.

## **C.10 Do you think your time estimating skills are improving?**

**Student #1:** From the chart, you know, I can see some improvement. But the problem is, I don't feel good. I mean if you give me a new project and estimate like the last few it is more complicated than before. Something like I don't feel like I'm improving, but from the chart it seems the last two projects are a little bit closer than the. But I don't think it reflects the reality because the last two projects is graphical interface. Actually its more complex than the first several projects. But I think that is just a coincidence it not really good and my ability has not improved.

**Student #4:** For me?

*Yes, you are they getting better?*

Yea, for the projects that I'm familiar with definitely. It is improving more and more accurate but, sometimes I'm getting to the new area it is bad.

*So basically, when your estimates are wrong it is because of a new thing that you are not sure about?*

Thats right. I can new thing. I never done before and it will take a lot of time to do it.

*So when your data does not match the next project is mostly when your estimates are...*

No, I think, you know, estimation only makes sense for, you know, because this whole idea is based on experience. If you don't have that it will fail.

**Student #5:** I think I am. Yea.

**Student #6:** Estimation, I think I'm not because of the last two projects were different from before. They were GUI projects so it is harder to estimate.

*So, before these two different ones, if you had a similar one do you think you are getting better?*

Yea, yea I think so.

*Ok, so its because you have a new thing.*

Yea, a different kind.

*you're not as good because you don't have experience doing that.*

Yea.

**Student #8:** Yes, yes definitely.

**Student #9:** Always the same.

*So, it is staying about the same?*

Yea, it is not bad, so Dr Johnson says is correct enough.

**Student #11:** Yea, I think so.

**Student #12:** Yea, I think so. Especially, when you estimate, you can adjust which part you are familiar. How much time adjust according the time I collect. Before how many line, how many times, adjust all of that.

**Student #13:** Not really, because it all depends on the project. You know, if I, like for the first of the project, it is hard to estimate. But, when you add something or copy of you design, you know, how many a basically how many lines of code you are going to add. So, at that time you can do a

best estimation, but if you like the first time you add sometime add a GUI something. That is a lot of code and sometimes you know you have to try different GUI. It is not really you write the code and it is not really. You fix the bug. it is not a bug. It is just you have a bad GUI it takes a long time. It is hard to estimate.

**Student #14:**

**Student #15:** I think that since from my data it is pretty good for me. The only thing I have a problem is sometimes it is very difficult for me to estimate the size of the method. If the method counting is wrong then my size estimate is different.

*But still within 20%?*

Yea, I feel that because the GUI has quite a more code than the others. I guess that is what I think. I found. So, I tried to raise the line of code per method. So tried to adjust that.

## **C.11 What caused your estimates to be wrong?**

**Student #1:** I cant think of one thing, there are actually have several problems. Every time I think the problem is different. Sometimes I just got the design very wrong. I forgot something or sometime I try to use big chunk of old code. Other times it just different. I try to think many things caused it to be wrong.

**Student #4:** Thats right. I can new thing I never done before and it will take a lot of time to do it.

**Student #5:** I think there is no way I can get it right.

*So its just impossible to get the right one or it is very very hard?*

Yea, it is hard.

**Student #6:** Wrong time, usually just because of problems of my design. I don't go to so details. I designed some methods but when I really write them, they are very long, especially in GUI. Need to set up a larger thing. It is hard to predict the size.

**Student #8:**

**Student #9:** If time estimate is wrong, my problem is because my size estimate is wrong.

*Ok, so because you get the size wrong you get the time wrong.*

Yea.

**Student #11:** I used a couple of excuses usually but, I don't know if that is really the case. I did a big refactoring but actually that refactoring took two hours and I was off by about 8 hour or something. I don't know what the excuse was for that one I just had to learn too much stuff.

**Student #12:** I think for me it happened usually class part. Sometimes, you know, ideas how I code. When you, when you, I think I use five classes for this one, two classes for the other one but something happened I had to use more classes for next one.

**Student #13:** You know, if like for the first of the project, it is hard to estimate. But, when you add something or copy of you design, you know how many, basically, how many lines of code you are going to add. So at that time you can do a best estimation. But, if you, like, the first time you add sometime, add a GUI something. That is a lot of code and sometimes, you know, you have to try different GUI. It is not really you write the code and it is not really you fix the bug. It is not a bug. It is just you have a bad GUI. It takes a long time it is hard to estimate.

**Student #14:** I think it might because now were working on GUI, right. So, I'm not quite familiar with it. So, when I implement it, I think this one might be OK. But, when I, you know, testing it, might not work. So, that I have to redo my work. So, sometimes I underestimate my testing time..

**Student #15:**

## **C.12 Did you compare your Leap data with other students in the class?**

**Student #1:** No, I did not.

**Student #4:** No.

**Student #5:** No.

**Student #6:** No.

**Student #8:** I never. I didn't have a chance yet.

**Student #9:** No.

**Student #11:** No.

**Student #12:** No.

**Student #13:** No.

**Student #14:** No.

**Student #15:** No.

### **C.13 Compare your programming skills now to your programming skills before the class.**

**Student #1:** Actually. I can't think. I don't know. It is hard to know my programming skills. They improved because in this class, actually Philip didn't really teach any programming or design patterns or patterning. I don't know. I really did one thing. I learned, I think the, how to pay attention to the phase I'm working on. I never thought of that before but, really I think the work. I think I didn't improve my programming skills, but maybe I did the old program works.

*By practice?*

Yea, I think so.

**Student #4:** I think for the area we are working on they seem to be better.

*In what ways are they better than before?*

I don't think that the, this benefit is from the Leap tool.

*But in the class?*

In the class, you know, for the final one, the editor one, you know, before you know I ?? method. Before, I did some similar project but it, you know, is a mess. At the end of the project, it turned to a mess because I made lots of menus a lot of buttons. The functions related to each other but this one because we designed. We make the design, well, at the end of the project it you know is organized.

**Student #5:** Much better.

*In what sort of ways are you much better?*

I think, at first, I don't know how to. I don't think it is because of leap. More familiar.

**Student #6:** Yea

*Are you better the same?*

I think I am much better.

*In what ways are you much better?*

Because the we using the programming styles. Before, I didn't I focus on the design phase and even on the implementation phase actual. I spend more time on design phase.

**Student #8:** In the beginning, I talked to Dr Johnson and I asked him if I should drop the class, because he said you need 2 semester of java programming skills and a experience and I don't have any. Because at the time when I took java 2 1/2 years ago. When java just came out java 1.0 something. He said if you don't mind you can drop the class and take it at another time but I decided to take a try. So in the beginning, I actual, I was, you know, I didn't know. But, now, you can give me a java program, I might not do the job professionally but I think I can do the job.

**Student #9:** Better, much better.

*What is better about your skill now then before?*

You mean programming skills. I think before this class I do the java programming. I not very familiar, but now I think I have data.

**Student #11:** I'd say I've improved if I didn't get better but I've learned.

*What ways have you improved as a programmer?*

I'd say I've gotten a lot more practice at Swing for one thing... lets see if I can put a few ideas into practice that I have not done before but thats pretty vague.

**Student #12:** You know, before the class, I not think too much about design, you know. You could estimate class. Think about time. How much lines. Just do something. But, now, just follow the Leap step by step you over and now the Leap. After you finish, you can get Leap data. We can analysis. So, something happens. So, the next time I estimate, I can adjust it. Thats kind of it is very hard.

**Student #13:** I don't know.

*You don't know?*

Because, you cannot just only look the size of LOC. Because, like for GUI, you have a lot of code. But you just, sometime, you just, what you did is just copy paste. So, I think it not really, that its very productive. Maybe, you project a in certain ways.

*Are you a better programmer now because of the class?*

Of course, I'm a better programmer, but that is because I practice I write the code.

*So just by just doing eighth projects what ever?*

For java, it is not really for other kinds of program languages. I think I am a better java programmer.

**Student #14:** I think they're much better

*Much better in what sort of way?*

I think I get to know the Java more and I can arrange the schedule and schedule the time for each phase accurately and I could I mean I could arrange my schedule well

**Student #15:** The scale is too far right. What time I think that, you know, one of the things that you know by practicing more, you, the java skill. You certainly can get some experience but I think record the defect type made me some impression. That, ok, the next time I want to have to forget that the defect. So, I think I like the defect recording because the next time, if you have it this time, the next time you probably not have it. Because, I have the impression this is the time I spent and this is what kind of error I recorded.

## **C.14 What is the most important thing you learned from this class?**

**Student #1:** Most important thing I think that I learned, you cannot just program randomly. You shouldn't just program. You some how, you should collect some data. You should learn something from your experience. You shouldn't just throw away every thing you wrote before. I know we should treat this professionally

**Student #4:** I think management, management. How to manage the project information and ah. I'm, I'm thinking to develop another tool to manage the employees because I when did a intern in telecom. They had a similar tool but it a database and web application, two layer. But, we used a

database and date/time object. We defined ourself and application layer and organized at a server, client, client-server application.

**Student #5:** Well, I think collecting defects is good. You find out what mistakes you make. Well, and I have a idea how to estimate time, although not too accurate. But I can get a range.

*And your estimating the range, do you think your, before this class were you able to estimate your time do you think you could?*

Before, I don't think I had much experience and so I think now I have enough data.

*So if you were going to say how many days do you think your estimate of how many days would be close?*

I think so maybe one or two days off.

**Student #6:** Most important thing. I think the most important thing is making, realizing the design, the importance of design and also the data recording. Sometimes, I really begin to know even the time data estimation is not very accurate but at least I have some idea how long it will take.

**Student #8:** Well, I don't know how describe them, quite a few things. One of the things, if you want to be a good programmer, you cannot just code. When someone gives you a project and, ok, I will start and spend some time. You don't have any feeling about how long it will take and the size it will end up with. That is one of the things that I ... I never. I think I have some intuition but there's no a real clear picture about it. So that is one of the things that has been changed. So, I think in the future, if the system can provide that kind of help, what ever the environment knows. It knows you're working. A better thing is all kinds of things, you know. That's also, I think, you did the right thing have a defect and put it down and, you know. That is kind of nice like but, before, you know, I think, when I work, I did some a little bit but in C or VB and sometimes the same kind of problem seems to happen. Oh, that has happened some time ago but this time I recorded everything because I still have a chance to go back look at them. Sometimes, you know, just refresh yourself don't get stuck with that. If you see one defect takes you 30 minutes. One time I got a logic thing not a pointer or anything just a logic problem and I just realized it took me like more than 20 minutes almost 30 min to fix it. So, when I look at it and say ok this kind of problem, the chance it will happen is kind of less. So those things, yea, quite a few. I don't think I can list everything in my mind but that is something that just happens, you know. Maybe you don't even know it.



**Student #9:** I think, first important, I think, is the improvement in java programming and I think the second is I think this is helpful. If, like, collect, to collect data. Then in programming I will pay attention to what is actual happening. Then try to forecast is much easier.

**Student #11:** I read about that extreme programming and that sounds interesting. What did I get from this class, I don't know, you know. I went from being completely helpless as far as estimation goes but now I have a few crutches to lean on. From lying down to being on crutches can't get up the stairs in either case.

**Student #12:** For this class, I think to be a software engineer.

**Student #13:** Hard to think because of all the questions about this class. I don't know, I know like, I know like my productivity how many line of code per hour. But, I don't know whether I should trust it or not. I don't know whether I should tell my boss I'm that productive or something.

**Student #14:** First is familiar with Java. Then I think I'm more familiar with how fast I can program because I used to be not sure how much time I spend on program. But now I think I can make an estimate and it is close to the actual time is spend.

**Student #15:** I think that, you know, that one thing it very interesting is a like software engineering. You learn and then you estimate your project based upon your experience and which is I like that because it is based upon your experience but not set on a mutual standard. Everybody on the same standard. I think different people have different learning curves and different development. I think this is a good thing and I said I like the defect recording and a time estimation I also liked.

## **C.15 What would you change about Leap to make it better?**

**Student #1:** You mean what aspect?

*What would you change about the leap toolkit to make it better?*

Automatic collection.

*Automatic collection? So it would know when you are coding and record the time for you?*

Yea.

**Student #4:** I in what areas anything?

*Your biggest complaint with leap this semester.*

The biggest issue, I'm not sure it is because of Leap or because of Java because it takes a lot of memory.

*Ok, so it...*

I don't know, how do you parse the string? I think parsing string took a lot of time and could be optimized.

*So, it took a lot of memory and it was slow?*

Yea, slowed down the whole machine.

*So, the whole machine slowed when you ran Leap?*

Thats right, usually if I run. If developing some java application on a relative slow machine, I would not start Leap because of this problem. But, for me personally, I like to run the timer every time but because for you know it depends on which machine I'm using.

**Student #5:** I think it should give more help. Thats it it should give more help.

**Student #6:** Change. I think change it. The, I think, when you edit something in the, the panel. I forgot what it is. When you right click the the projects.

*Thats Hee.*

Yea, so if we condense data we can see the, the Leap data. If it is really a lot of data from the sometimes...

**Student #8:** Definitely you shouldn't take too much time to add the side on a defect. Because, when you are doing programming you don't want to spend too much time on just limited to Leap. The other thing is for the bad experience that I had in the beginning is you know sometimes when I finish well everything. I know just do postmortem, you know, condense data. That as a new user, I don't have that problem any more but in the beginning it is, it might get some of the new users. Just because, ok, ok, that is enough and I don't want to do it any more. So, there is, there is a user. It is not sure. It is, it is a little bit hard to imagine what I felt at that time. But, I do feel that if I spend more than one hour on the Leap thing, I would feel hopelessly irritated. But, for now I don't feel that anymore. I mean just, ok, I'll, there is not much tricky there and everything is intuitive. If you don't know anything you just play a little bit on. I don't feel that anymore. That is something that for a new user that might get in his or her way. Might push them away. So, other things, I would

prefer that something in a special case. I would prefer that Leap had the has some, you know, force people to use that to record time. When you do things not write down and put it in. Leap is very neutral on that. What ever you do fine. But, I think people should, who ever is serious about this should use that because that would save time. I think that is what Dr Johnson is doing but Leap doesn't force that. It, just, I would just let the user use Io on the screen just click.

**Student #9:** I think it is a Java features, because like yesterday, I used Leap, it get many times. I think it is Java. It is really slow.

*So it is slow. It could be faster?*

Yea.

**Student #11:** Oh, I'd make it, you know, so you never have to do anything consciously. Obviously, its painful like, ok, I'm starting now. Hello, let me open up the tool. Let me then load this file. Fiddle around with a few things, then oh, what was I doing again? What else, its hard for me to separate, you know, the particular process you are using from the tool, because supposedly the tool is not tied to that process. But Philip is sort of asking us to use the process and he supported it by giving us the definitions and everything. So we were using it. But, I was not always real comfortable with it. Particularly, I would like to have, you know, sort of smaller things that we could estimate. And control of the grainsize or have something in the process were I might make a big vague estimate that wasn't perfect. Big piece and then cut that into little classes. Make them more specific estimation about that. But not have to them all, you know. All the estimation before is, start doing the work, the first chunk is ok, figure out what the heck is going on then you could estimate that chunk and then do it something like that. I'm thinking something really vague.

*Does that lead to something like a tracking, heres my big goal it is supposed to have threes smaller pieces, I've finish this piece and and this one was quicker than I guessed. Is that what you are looking at? Some way of breaking this bigger task with a fuzzy estimate into smaller tasks or combining smaller tasks into bigger tasks?*

Having correcting the big one as you are go along, when you are half way through you say, "oh yea this one is big damn I screwed up this is never going to be done on time." Then you can change the big one. give or take.

**Student #12:** Think about. When I do the Leap I feel one thing it is slow. I think it is a Java problem not a Leap problem.

**Student #13:** The idea of leap is very very good. I think you should just fix the bugs and I don't know whether it is good to use Java. Java is always slow you can do nothing about it. Like when I start it, I always have to wait awhile before it comes up before slow my machine.

**Student #14:** I think because maybe its made by Java. So sometimes its kind of slow, but also I think Java itself has some bugs. So when you make a program, you cannot avoid that. ... I think it looks good but you could maybe, because I think its not very convenient when I input the time data into Naia and ...

*So maybe a better way of inputting time?*

Yea.

**Student #15:** I think it, your interface the Leap interface is pretty good. The only thing that bothers me is that it crash my file and I think that yea Leap interface is ok.

*How about performance is it too slow enough fast enough?*

Not fast enough, and that is a Java problem. They call it a just in time compiler it that true or not?

## **C.16 In what situations would you use Leap in the future?**

**Student #1:** I think I should use it lots of time when I program. I think I feel some pressure when I use leap, but I think I don't use it all the time because too much stress. Maybe sometimes, I do some curious programming I don't use Leap, because that is just casual. But, when I want to learn something I will use Leap.

**Student #4:** I will use it to, you know, to count my programming the Java. I definitely will but for others, I hope you guys can add some other you know counters for visual basic or c++. I will use it but obviously, you know, let me know your new release and I will download it and use it. It is perfect.

**Student #5:** Well, just when do one of kind of projects like DB project recording how long it takes.

**Student #6:** I don't know. I do want to use Leap but, I don't want to record so much data. I just want to record the size and time. Yea, and maybe some some really serious defects. Maybe, I want to record them, but most of the data I don't want to record.

**Student #8:** Programming for sure and other things. I do not have much feeling to what I do. Technical work of just regular papers so I cannot write it critically now. But if I have a chance, the situation that I would use leap in, if I know I would work in this environment on this similar kind of project track and I would use it. But other wise, for just for one time or even several times and I, then I wouldn't use it. Ok, the other thing is if my boss has a good sense of time or size, then definitely I would. I mean you have no other way and this is actually a great help. But, if my boss never tells he says just do it, then somehow I don't know. I cannot predict. But, for my self benefit I would use it. But, if it doesn't make any difference in my working environment then I might, you know. It is extra effort for my.. so there are a few things that I really don't know at this time.

**Student #9:** I think many areas maybe. If you were not fit all the requirements like size, time, defect. Maybe I would use just one feature.

**Student #11:** Well, I guess for Java development. I'm really having some hardship for this other project I'm doing. Trying to do a some paper grading. I have two final assignments for 111 that I have to grade. I don't know like I can't get into it. Right now, I'm doing and just writing down the times. I'm going to figure out the phases and stuff. I'm going to plan it after it is over I guess. I'm collecting data I'm not going to find any uses. I have to be done by Thursday so there is time pressure.

**Student #12:** Oh, I think if I do something programming job from my computer. I think I would use it. So, Leap record to collect data for very good for the future.

**Student #13:** I think I use leap for by myself or when I feel I can know my productivity when I ...

**Student #14:** Not really.

*What would make you want to sort of want to use leap, or what kinds of situations would make you?*

I think if I make programs maybe I would use Leap but on the other stuff I might not use it. I think it is helpful when you program but on other circumstances I'm not sure.

**Student #15:** I think if I'm using I will use it for the job probably, the size. Actually, you know that when we had module 14 to customize, we have to estimate the size. If you can provide the tool to estimate the size for any computer project, then I can use it. But for the Java right now you can use Java for the programming. Then I probably for sure I'll probably use it because of my project data. I can estimate the project. At least give me an idea of how long for this project. That would be good.

# Bibliography

- [1] Jody Armour and Watts S. Humphrey. Software product liability. Technical Report CMU/SEI-93-TR-13, Software Engineering Institute, Carnegie Mellon University, August 1993.
- [2] Robert D. Austin. *Measuring and managing performance in organizations*. Dorset House, 1996.
- [3] Peter M. Blau. *The Dynamics of Bureaucracy: A Study of Interpersonal Relations in Two Government Agencies*. The University of Chicago Press, 2nd edition, 1963.
- [4] Anne M. Disney. Data quality problems in the Personal Software Process. M.S. thesis, University of Hawaii, August 1998.
- [5] Anne M. Disney and Philip M. Johnson. Investigating data quality problems in the PSP. In *Proceedings of the ACM SIGSOFT Sixth International Symposium on the Foundations of Software Engineering*, pages 143–152, Lake Buena Vista, FL, November 1998.
- [6] Robert H. Dunn. *Software Quality: Concepts and Plans*. Prentice Hall, 1990.
- [7] Khaled El Emam, Barry Shostak, and Nazim Madhavji. Implementing concepts from the Personal Software Process in an industrial setting. In *Proceedings of the Fourth International Conference on the Software Process*, Brighton, England, December 1996.
- [8] Michael E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211, 1976.
- [9] Pat Ferguson, Watts S. Humphrey, Soheil Khajenoori, Susan Macke, and Annette Matvya. Introducing the Personal Software Process: Three industry cases. *IEEE Computer*, 30(5):24–31, May 1997.
- [10] The WWW formal technical review archive. <http://www.ics.hawaii.edu/johnson/FTR/>.

- [11] Robert L. Glass. *Software Runaways: Lessons Learned from Massive Software Project Failures*. Prentice Hall, 1998.
- [12] GNU emacs - GNU project - free software foundation(FSF). <http://www.gnu.org/software/emacs/emacs.html>.
- [13] B. Hall. Participatory research: An approach for change. *Convergence: An International Journal for Adult Education*, 8(2):24–31, 1975.
- [14] Will Hayes and James W. Over. The Personal Software Process (PSP): An empirical study of the impact of PSP on individual engineers. Technical Report CMU/SEI-97-TR-001, Software Engineering Institute, Pittsburgh, PA., 1997.
- [15] Joel Henry. Personal Software Process studio. <http://www-cs.etsu.edu/softeng/psp/>, 1997.
- [16] M. Hult and S. Lennung. Towards a definition of action research: A note and bibliography. *The Journal of Management Studies*, 17:241–250, 1980.
- [17] Watts S. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley, January 1995.
- [18] Watts S. Humphrey. Using a defined and measured Personal Software Process. *IEEE Software*, 13(3):77–88, May 1996.
- [19] International Organization for Standardization. *ISO Standards Compendium - ISO 9000 Quality Management*, 7th edition, 1998.
- [20] Philip M. Johnson. An instrumented approach to improving software quality through formal technical review. In *Proceedings of the 16th International Conference on Software Engineering*, pages 113–122, Sorrento, Italy, May 1994.
- [21] Philip M. Johnson. Supporting technology transfer of formal technical review through a computer supported collaborative review system. In *Proceedings of the Fourth International Conference on Software Quality*, Reston, VA., October 1994.
- [22] Philip M. Johnson. Design for instrumentation: High quality measurement of formal technical review. *Software Quality Journal*, 1995.
- [23] Philip M. Johnson. Measurement dysfunction in formal technical review. Technical Report ICS-TR-96-16, Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii 96822, November 1996.



- [24] Philip M. Johnson and Danu Tjahjono. Improving software quality through computer supported collaborative review. In *Proceedings of the Third European Conference on Computer Supported Cooperative Work*, September 1993.
- [25] Philip M. Johnson, Danu Tjahjono, Dadong Wan, and Robert Brewer. Experiences with CSRS: An instrumented software review environment. In *Proceedings of the Pacific Northwest Software Quality Conference*, Portland, OR., 1993.
- [26] S. Khajenoori and I. Hirmanpour. An experiential report on the implications of the Personal Software Process for software quality improvement. In *Proceedings of the Fifth International Conference on Software Quality*, pages 303–312, October 1995.
- [27] Nancy G. Leveson and Clark S. Turner. An investigation of the Therac-25 accidents. *IEEE Computer*, 1993.
- [28] The locc system. <http://csdl.ics.hawaii.edu/Tools/LOCC/LOCC.html>.
- [29] Carleton A. Moore. Supporting authoring and learning in a collaborative hypertext system: The Annotated Egret Navigator. M.S. Thesis Proposal ICS-TR-94-16, Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii 96822, December 1994.
- [30] Mark C. Paulk, Charles V. Weber, Bill Curtis, and Mary Beth Chrissis. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, 1995.
- [31] PSP resources page at the University of Karlsruhe. <http://www.wipd.ira.uka.de/~gramberg/PSP/>.
- [32] PSPtool version 0.6. <http://www.virtual.net.au/simtqc/description.html>.
- [33] M. Ramsey. Experiences teaching the Personal Software Process in academia and industry. In *Proceedings of the 1996 SEPG Conference*, 1996.
- [34] Khalid Sherdil and Nazim H. Madhavji. Human-oriented improvement in the software process. In *Proceedings of the 5th European Workshop on Software Process Technology*, October 1996.
- [35] Barry Shostak. Adapting the Personal Software Process to industry. *Software Process Newsletter* #5, Winter 1996.
- [36] Timetracker - an x-windows timekeeper. <http://www.alvestrand.no/domen/titrax/TimeTraker.html>.

- [37] Danu Tjahjono. Evaluating the cost-effectiveness of formal technical review factors. Ph.D. Dissertation Proposal. CSDL-TR-94-07, University of Hawaii, Department of Information and Computer Sciences, 1994.
- [38] Danu Tjahjono. Comparing the cost effectiveness of group synchronous review method and individual asynchronous review method using CSRS: Results of pilot study. Technical Report ICS-TR-95-07, Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii 96822, January 1995.
- [39] Danu Tjahjono and Philip M. Johnson. FTArm demonstration guide (version 1.2.0). Technical Report ICS-TR-95-19, Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii 96822, October 1995.
- [40] Danu Tjahjono and Philip M. Johnson. FTArm user's guide (version 1.2.0). Technical Report ICS-TR-95-18, Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii 96822, October 1995.
- [41] Dadong Wan and Philip M. Johnson. Experiences with CLARE: a computer-supported collaborative learning environment. *International Journal of Human-Computer Systems*, 41:851–879, December 1994.
- [42] XEmacs: The next generation of emacs. <http://www.xemacs.org/>.
- [43] Robert K. Yin. *Case Study Research Design and Methods*. Sage Publications, Inc., second edition, 1994.
- [44] Edward Yourdon. *Structured Walkthroughs*. Prentice-Hall, 1979.