Software Process Control Without Calibration

Oussama El-Rawas

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science in Electrical Engineering
(with Software Engineering as a major focus)

Tim Menzies, Ph.D., Chair
Hany Ammar, Ph.D.
Bojan Cukic, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2008

Keywords: Search-based Software Engineering, COCOMO, COQUALMO,
SCHEDULE, THREAT, Simulated Annealing, STAR, Managing Uncertainty, AI Search

Abstract

Software Process Control Without Calibration

Oussama El-Rawas

Software process control is important for large enterprise since it is essential for software project management. Boehm [10, 12–15, 20] argues that the best way to do software process control is reusing old proven models (e.g. COCOMO for effort, COQUALMO for defects, etc) while tuning them to local data in order to obtain accurate estimates. This however suggests that historic data is available related to the use of these models in previous software projects. This is not the case, as the availability of relevant historic data related to the use of the above models in a specific software environment is scarce, whether due to the lack of documentation or the unwillingness of companies to disclose such data [63].

To bypass this problem, we implemented a system called STAR. This system uses a combination of an AI search algorithm and a back-select algorithm to determine recommended work that needs to be done on a software project. STAR also has the ability to use multiple models in the evaluation of recommended practice; a feature that is not available in any previous work to the best of our knowledge. The models used are part of the USC family of software engineering models [15] and include: COCOMO II for effort, COQUALMO for defects, a schedule model for development time, and the Madachy [55] threat model for risk assessment.

Upon implementing STAR, we observed stable results that were comparable to those generated by tools currently used, while bypassing the local tuning problem that those tools face. In addition, we were able to tackle several issues related to software process control using STAR. So, in the future we recommend that, in situations where local tuning data isn't available, we exploit the uncertainty of not having local tuning data by searching for stable conclusions withing the space of possible recommendations using AI search engines similar to STAR.

Dedication

I would like to dedicate this work God, who has seen me through many trials and has given me the ability and the talent to produce this work, and who continues to show me grace everyday through his son Jesus Christ.

Acknowledgments

I acknowledge my parents, whom without their unwavering support I would not have been able to achieve what I have; my brother whom I dearly love; my friends and family here in the US and in Morgantown, WV specifically: I've been blessed by knowing you and by how you've embraced me as one of your own; my research advisor Dr. Tim Menzies for guiding my research and keeping me on track with what I need to do; my committee who have each contributed towards this work in one way or another; the ICSE 2008 May 13 panel members, facilitators and scribe (thank you Greg Gay); all my colleagues that I've had the pleasure to work with, who include Dan Baker, Omid Jalali, Phillip Green, Steve Williams, among others; Julian Richardson and Jarius Hihn for supporting this work either through funding or by supplying valuable data and feedback; and last but not least my roommates for being patient with me and Amanda Berousek for support and encouragement.

Contents

List of Figures

Chapter 1

Introduction

Software process control allows managers to control software projects and keeping them within required specifications, whether for performance or cost or time to market. A repeated theme in literature [10, 12–15, 20] suggests that the most effective way of doing this is through using software engineering models that have been pre-calibrated to the particular software development environment. This assumes that there is an abundance of historical information that is available to tune these models. This however is a false assumption, as in reality there is a "data drought" regarding project data [60], causing uncertainty within the software models. So instead we developed a system that uses this uncertainty and mutates the models around the ranges of possible tunings using an AI search engine, while at the same time evaluating possible recommendations to control the software project.

1.1 Motivation

Process models have many purposes including estimating project parameters or conducting what-if queries to find better ways to organize a project. Most process models calibrate their internal settings using local data. Collecting this data is expensive, tedious, and often an incomplete process

[63]. For example, after 26 years of trying, Boehm et al. [60] have only collected less than 200 sample projects for the COCOMO database. Also, even after two years of effort, only 7 records were added to a NASA-wide software cost metrics repository. For this reason we wanted to investigate whether or not we had the ability to:

1. Make process decisions without historical data while achieving stability in those decisions.

2. Produce accurate estimates of different project properties (such as effort, defects, etc).

3. Move on to investigate other issues related to software process control.

We needed to determine which SE models to use for our studies, for which we chose the USC family of models. This includes the the COCOMO II.2000 effort model [15, p254-268], the COQUALMO defect model [15, p29-57], the Madachy risk model [15, 284-291], and the COCOMO II based Months time of development (SCHEDULE) model [15]. These were chosen for their openness in terms of specification, their widespread adoption by such institutions like NASA and JPL, and for the wider availability of data on these models compared to other models in the software engineering industry. This is useful to us since it helps us determine variability in the parameters of these models, and therefore the space of tunings.

Having determined the models to be used, we attempted to evaluate whether we could fulfill the above stated goals. By using an AI search engine to explore the range of tunings of the selected process models, we were able to produce results with a good deal of stability, while at the same time achieving control over project estimates. We were also able to produce estimates with the ball park of figures produced with tuned models. In addition, we were able to use this system to do other studies relating to the subject of software process control.

Based on our results and empirical data, we present the following statement of thesis.

1.2 Statement of Thesis

Variability in software process model output arises from (a) uncertainty in model inputs relating to the project and (b) uncertainty in the tuning of internal parameters that control the process model outputs. We find that, for USC process models such as COCOMO and COQUALMO, the model ouputs are less dependent on the uncertainty in the internal parameters of the model as compared to the dependence on the uncertainty of the project parameters. We are hence able to control the variance of the process models by setting project parameters rather than resorting to calibrating the models.

1.3 Contribution of this Work

The area of search based software engineering is one that is growing at the moment at a quick pace [40], and this work further contributes to this field. We introduce a novel method of evaluating and controlling the software process, using four of the USC family models in conjunction with an AI search engine, without the need for local tuning. From a business perspective, this result means that certain process models can be used for decision making in one of two ways:

1. Either constrain the tuning variance using historical data;

2. Or constrain the project choices using an AI search engine like STAR, presented in §3.3.

Note that this second method avoids a lengthy and expensive data collection phase prior to decision making. This result is of tremendous practical benefit since it is often very difficult to find relevant data within a single organization to precisely tune all the internal parameters inside a process model. This thesis, in addition to the above, also contributes several case studies and experiments. The data of these studies is publicly available, along with the tool developed and its source code, for third parties to attempt to regenerate the results presented. Note that this work has also been used as a proof of concept for further work.

1.3.1 Publications from this work

This work has generated several publications:

- **ICSE'09**: *How to avoid drastic software process change*. [64]
- **ASE'07**: *The business case for automated software engineering*. [60]
- **ASE'07 workshop**: *On the value of stochastic abduction*. In *International Workshop on Living with Uncertainty* . [58]
- **ICSP'08**: *Accurate estimates without calibration* (in the International Conference on Software Process). [59]

In addition to the above publications, there are plans to generate several others directly derived from this work.

1.4 Relation to Other Work

This thesis discusses the design of an AI decision support system called STAR. STAR's design has inspired other work by other WVU graduate students: For example, the author assisted another WVU graduate student, Mr. Steve Williams, in his work building a LISP version of STAR.

In order to assign appropriate credit to the work reported in this document, it should be noted that:

- For all the STAR results reported in this thesis, El-Rawas was the lead analyst/programmer.
- For all the SEESAW results reported in this thesis, Williams was the lead analyst/programmer while El-Rawas served as programmer and assistant.

1.5 Document Description

The rest of this document is organized as follows. Chapter 2 will present previous work that is relevant to this thesis. This is followed by a detailed description and analysis of STAR in chapter 3, accompanied several internal studies to verify its use. Chapter 4 presents the NASA case studies and the case for the use of automated software engineering tools. Chapter 5 presents studies done in conjunction with experts in the field. Chapters 6 and 7 are case studies into two software engineering practices: Drastic change and "better, faster, cheaper". The document is concluded with some notes on the results, external validity and future work that is already ongoing. The appendix contains the code listing and how to obtain the tool presented here.

Chapter 2

Related Work

To motivate this work, we discuss a model with a widely-used software engineering predictor for development effort, threats, development time, and defects.We identify certain problems with this model (calibration instability) which makes it inadequate to report point solutions. Hence, this chapter reviews methods for using AI search over the space of possible solutions.

Note that this thesis explores one such method and other WVU graduate researchers ([64], [71], [34]) are exploring others. In the following, all references to STAR describe the code developed for this thesis and all references to SEESAW describe an alternate system developed by Menzies and Williams [64] that follows the design principles described in this thesis.

2.1 Models

In this section we will present the USC software engineering models that were used in this work and which influenced the fitness function used.

	Definition	Low-end = {1,2}	Medium ={3,4}	High-end= {5,6}
Defect removal features				
execution-based testing (ett)	all procedures and tools used for testing	none	basic testing at unit/ integration/ systems level; basic test data management	advanced test oracles, assertion checking, model-based testing
automated analysis (aa)	e.g. code analyzers, consistency and traceability checkers, etc	syntax checking with compiler	Compiler extensions for static code analysis, Basic requirements and design consistency, traceability checking.	formalized specification and verification, model checking, symbolic execution, pre/post condition checks
peer reviews (peer)	all peer group review activities	none	well-defined sequence of preparation, informal assignment of reviewer roles, minimal follow-up	formal roles plus extensive review checklists/ root cause analysis, continual reviews, statistical process control, user involvement integrated with life cycle
Scale factors:				
flex	development flexibility	development process rigorously defined	some guidelines, which can be relaxed	only general goals defined
pmat	process maturity	CMM level 1	CMM level 3	CMM level 5
prec	precedentedness	we have never built this kind of software before	somewhat new	thoroughly familiar
resl	architecture or risk resolution	few interfaces defined or few risk eliminated	most interfaces defined or most risks eliminated	all interfaces defined or all risks eliminated
team	team cohesion	very difficult interactions	basically co-operative	seamless interactions
Effort multipliers				
acap	analyst capability	worst 15%	55%	best 10%
aexp	applications experience	2 months	1 year	6 years
cplx	product complexity	e.g. simple read/write statements	e.g. use of simple interface widgets	e.g. performance-critical embedded systems
data	database size (DB bytes/SLOC)	10	100	1000
docu	documentation	many life-cycle phases not documented		extensive reporting for each life-cycle phase
ltex	language and tool-set experience	2 months	1 year	6 years
pcap	programmer capability	worst 15%	55%	best 10%
pcon	personnel continuity (% turnover per year)	48%	12%	3%
plex	platform experience	2 months	1 year	6 years
pvol	platform volatility $(\frac{frequency\ of\ major\ changes}{frequency\ of\ minor\ changes})$	$\frac{12\ months}{1\ month}$	$\frac{6\ months}{2\ weeks}$	$\frac{2\ weeks}{2\ days}$
rely	required reliability	errors are slight inconvenience	errors are easily recoverable	errors can risk human life
ruse	required reuse	none	multiple program	multiple product lines
sced	dictated development schedule	deadlines moved to 75% of the original estimate	no change	deadlines moved back to 160% of original estimate
site	multi-site development	some contact: phone, mail	some email	interactive multi-media
stor	required % of available RAM	N/A	50%	95%
time	required % of available CPU	N/A	50%	95%
tool	use of software tools	edit,code,debug		integrated with life cycle

Figure 2.1: Features in the USC software process models.

2.1.1 Effort - COCOMO II

The COCOMO II model [15] is and evolution of the original COCOMO 81 model [11] which was published by Barry Boehm. Both of these models have the same form referred to in Equation 2.1,

	vl	l	n	h	vh	xh
Scale factors:						
flex	5.07	4.05	3.04	2.03	1.01	
pmat	7.80	6.24	4.68	3.12	1.56	
prec	6.20	4.96	3.72	2.48	1.24	
resl	7.07	5.65	4.24	2.83	1.41	
team	5.48	4.38	3.29	2.19	1.01	
Effort multipliers:						
acap	1.42	1.19	1.00	0.85	0.71	
aexp	1.22	1.10	1.00	0.88	0.81	
cplx	0.73	0.87	1.00	1.17	1.34	1.74
data		0.90	1.00	1.14	1.28	
docu	0.81	0.91	1.00	1.11	1.23	
ltex	1.20	1.09	1.00	0.91	0.84	
pcap	1.34	1.15	1.00	0.88	0.76	
pcon	1.29	1.12	1.00	0.90	0.81	
plex	1.19	1.09	1.00	0.91	0.85	
pvol		0.87	1.00	1.15	1.30	
rely	0.82	0.92	1.00	1.10	1.26	
ruse		0.95	1.00	1.07	1.15	1.24
sced	1.43	1.14	1.00	1.00	1.00	
site	1.22	1.09	1.00	0.93	0.86	0.80
stor			1.00	1.05	1.17	1.46
time			1.00	1.11	1.29	1.63
tool	1.17	1.09	1.00	0.90	0.78	

Figure 2.2: Co-efficient values according to the precalibrated COCOMO II.2000 model.

where c_i is the sum of the cost drivers and A is domain specific.

$$Effort = A * (c_i)(Size)^{Exponent} \tag{2.1}$$

The difference is in what *Size* and *Exponent* represent. In COCOMO 81, *Size* is dependent on deliverable source lines of code (DSI), while for COCOMO II it represents source lines of code (SLOC), function points (FP) and object points. In addition, the *Exponent* in COCOMO 81 is actually a fixed constant that is selected as a function of development mode, which only exists in COCOMO 81, while in COCOMO II it is dependent on scale factors, which were introduced in COCOMO II. This is evident in Equation 2.2. Other differences, as well as a guide to converting from COCOMO 81 to COCOMO II, are further elaborated on by D.J. Reifer and B. Boehm [74]. For this thesis, we will be exclusively using COCOMO II.

The COCOMO effort predictor estimates development effort in terms of person months where

one month is 152 hours (and includes development and management hours). In COCOMO, the *scale factors* SF_i of 2.1 effect effort exponentially (on KSLOC, i.e. Thousand Source Lines of Code) while *effort multipliers* EM_j effect effort linearly:

$$PM = a * \left(KSLOC^{\left(b+0.01*\sum_{i=1}^{5} SF_i\right)} \right) * \left(\prod_{j=1}^{17} EM_j \right) \tag{2.2}$$

where KSLOC is estimated directly or computed from a function point analysis; SF_i and EM_j are the *scale factors*; and *effort multipliers* respectively of Figure 2.1; and a and b are domain-specific parameters. In our NASA data, the following ranges were seen for a and b

$$(3.72 \leq a \leq 9.18) \wedge (0.88 \leq b \leq 1.09) \tag{2.3}$$

With the effort multipliers, off-nominal ranges (i.e. {vl=1, l=2, h=4, vh=5, xh=6}) change the prediction by some amount. In the special case of the nominal range (i.e. {n=3}), the factor it translates to is one; i.e. the nominal range make no change to the prediction. Hence, these ranges can be modeled as a straight line $y = mx + b$ passing through the point $\{x, y\}$=$\{3, 1\}$. Such a line has a y-intercept of $b = 1 - 3m$. Substituting this value of b into $y = mx + b$ yields:

$$\forall x \in \{1..6\}\ EM_i = m_a(x-3) + 1 \tag{2.4}$$

where m_a denotes the effect of effort multiplier a on *effort*.

The effort multipliers form into two sets:

1. The *positive effort EM* features, with slopes m_a^+, that are proportional to effort. These features are: cplx, data, docu, pvol, rely, ruse, stor, and time.

2. The *negative effort EM* features, with slopes m_a^-, are inversely proportional to effort. These features are acap, apex, ltex, pcap, pcon, plex, sced , site, tool.

Based on prior work [12], we can describe the space of known tunings for COCOMO effort mul-

tipliers to be

$$\left(0.073 \leq m_a^+ \leq 0.21\right) \wedge \left(-0.178 \leq m_a^- \leq -0.078\right) \tag{2.5}$$

Similarly, using experience from 161 projects [12], we can say that the space of known tunings for the COCOMO scale factors (prec, flex, resl,team, pmat) are:

$$\forall x \in \{1..6\}\ SF_i = m_b(x-6) \wedge (-1.56 \leq m_b \leq -1.014) \tag{2.6}$$

where m_b denotes the effect of scale factor b on *effort*.

Note that the above ranges for the slopes were obtained by finding the average slope for each COCOMO attribute for both effort multipliers and scale factors over the range of values of that attribute.

2.1.2 Schedule

The COCOMO schedule predictor estimates development time in terms of elapsed calendar months. This model is largely based on the COCOMO effort in that it uses those attributes, there values, and the calibrated coefficients (i.e, a and b). The following are the equations for calculating months:

$$\begin{aligned} F &= d + 0.2 * \left(0.01 * \left(\textstyle\sum_{i=1}^{5} SF_i\right)\right) \\ TDEV &= \left[c * (PM_{NS})^F\right] * SCED\% \div 100 \end{aligned} \tag{2.7}$$

where PM_{NS} is the effort calculated without the inclusion of SCED, SF_i are the *scale factors* of 2.1, c and d are domain-specific parameters that can be calibrated, and $SCED\%$ is the percentage of schedule relative to the nominal schedule.

2.3 shows the value of this parameter based on SCED. $TDEV$ is the development time in months calculated by this model, and F is referred to as the scaling exponent for schedule.

SCED	SCED%
1	75
2	85
3	100
4	130
5	160

Figure 2.3: The values of SCED%

2.1.3 Defects - COQUALMO

The COQUALMO model is used to estimate the amount if defects within a software project. COQUALMO has two core models, used three ways:

- The *defect introduction* model is similar to 2.2; i.e. settings to 2.1's effort multipliers and scale factors map to predictions about number of defects.
- The *defect removal* model represents how various tasks (peer review, execution-based testing, and automated analysis) decrease the introduction of defects.
- The above two models are repeated three times for defects introduction & removal for requirements, design, or coding.

COQUALMO follows the same convention as COCOMO for the effort multipliers; i.e. nominal values ($n = 3$) add nothing to the predicted number of defects. As the above, COQUALMO is:

$$\forall x \in \{1..6\}\ EM_i = m_c(x-3)+1 \tag{2.8}$$

where m_c denotes the effect of c on *defect introduction*.

The effort multipliers and scale factors form two sets:

1. The *positive defect* features, with slopes m_c^+, that are proportional to the estimated introduced defects. These features are flex, data, ruse, cplx, time, stor, and pvol.
2. The *negative defect* features, with slopes m_c^-, that are inversely proportional to the estimated introduced defects. These features are acap, pcap, pcon, apex, plex, ltex, tool, site, sced,

	rely	data	ruse	docu	cplx	time	stor	pvol	acap	pcap	pcon	aexp	plex	ltex	tool	site	sced
requirements:																	
xh			1.05		1.32	1.08	1.08	1.16								0.83	
vh	0.7	1.07	1.03	0.86	1.21	1.05	1.05	1.1	0.75	1	0.82	0.81	0.9	0.93	0.92	0.89	0.85
h	0.85	1.04	1.02	0.93	1.1	1.03	1.03	1.05	0.87	1	0.91	0.91	0.95	0.97	0.96	0.95	0.92
n	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
l	1.22	0.93	0.95	1.08	0.88			0.86	1.17	1	1.11	1.12	1.05	1.04	1.05	1.1	1.09
vl	1.43			1.16	0.76				1.33	1	1.22	1.24	1.11	1.07	1.09	1.2	1.18
design:																	
xh			1.02		1.41	1.2	1.18	1.2								0.83	
vh	0.69	1.1	1.01	0.85	1.27	1.13	1.12	1.13	0.83	0.85	0.8	0.82	0.86	0.88	0.91	0.89	0.84
h	0.85	1.05	1	0.93	1.13	1.06	1.06	1.06	0.91	0.93	0.9	0.91	0.93	0.91	0.96	0.95	0.92
n	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
l	1.23	0.91	0.98	1.09	0.86			0.83	1.1	1.09	1.13	1.11	1.09	1.07	1.05	1.1	1.1
vl	1.45			1.18	0.71				1.2	1.17	1.25	1.22	1.17	1.13	1.1	1.2	1.19
coding:																	
xh			1.02		1.41	1.2	1.15	1.22								0.85	
vh	0.69	1.1	1.01	0.85	1.27	1.13	1.1	1.15	0.9	0.76	0.77	0.88	0.86	0.82	0.8	0.9	0.84
h	0.85	1.05	1	0.92	1.13	1.06	1.05	1.08	0.95	0.88	0.88	0.94	0.94	0.91	0.9	0.95	0.92
n	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
l	1.23	0.91	0.98	1.09	0.86			0.82	1.05	1.16	1.15	1.07	1.08	1.11	1.13	1.09	1.1
vl	1.45			1.18	0.71				1.11	1.32	1.3	1.13	1.16	1.22	1.25	1.18	1.19

Figure 2.4: COQUALMO: effort multipliers and defect introduction

	prec	flex	resl	team	pmat
requirements:					
xh	0.7	1	0.76	0.75	0.73
vh	0.84	1	0.87	0.87	0.85
h	0.92	1	0.94	0.94	0.93
n	1	1	1	1	1
l	1.22	1	1.16	1.17	1.19
vl	1.43	1	1.32	1.34	1.38
design:					
xh	0.75	1	0.7	0.8	0.61
vh	0.87	1	0.84	0.9	0.78
h	0.94	1	0.92	0.95	0.89
n	1	1	1	1	1
l	1.17	1	1.22	1.13	1.33
vl	1.34	1	1.43	1.26	1.65
coding:					
xh	0.81	1	0.71	0.86	0.63
vh	0.9	1	0.84	0.92	0.79
h	0.95	1	0.92	0.96	0.9
n	1	1	1	1	1
l	1.12	1	1.21	1.09	1.3
vl	1.24	1	1.41	1.18	1.58

Figure 2.5: COQUALMO: scale factors and defect introduction

prec, resl, team, pmat, rely, and docu.

```
function defectsIntroduced() {
 return 10*Ksloc()*defectsIntroduced1("requirements") +
        20*Ksloc()*defectsIntroduced1("design") +
        30*Ksloc()*defectsIntroduced1("coding") }

function defectsIntroduced1(table) {
  # return the product of the Figure 2.4 and
  # and the Figure 2.5 figures  }
```

Figure 2.6: COQUALMO: defects introduced.

	automated analysis	peer reviews	execution_testing _and_tools
requirements:			
xh	0.4	0.7	0.6
vh	0.34	0.58	0.57
h	0.27	0.5	0.5
n	0.1	0.4	0.4
l	0	0.25	0.23
vl	0	0	0
design:			
xh	0.5	0.78	0.7
vh	0.44	0.7	0.65
h	0.28	0.54	0.54
n	0.13	0.4	0.43
l	0	0.28	0.23
vl	0	0	0
coding:			
xh	0.55	0.83	0.88
vh	0.48	0.73	0.78
h	0.3	0.6	0.69
n	0.2	0.48	0.58
l	0.1	0.3	0.38
vl	0	0	0

Figure 2.7: COQUALMO: defect removal

```
function Total_defects() {
  return defects("requirements",Coqualr) +
         defects("design",      Coquald) +
         defects("coding",      Coqualc)
}

function defects(what,table) {
  introduced     = defectsIntroduced1(what,table);
  percentRemoved = defectsRemovedRatio(what);
  return percentRemoved*introduced
}
```

Figure 2.8: COQUALMO: defects added and removed

```
function defectsRemovedRatio(table, auto,review,tool) {
  return (1 - drf(table,"automated_analysis")) *
         (1 - drf(table,"peer_reviews")) *
         (1 - drf(table,"execution_testing_and_tools"))
}

function drf(table,x ) {
  # return x's value in table from Figure 2.7
}
```

Figure 2.9: COQUALMO: ratio of defects removed

The space of tunings for defect introducing features are:

$$
\begin{aligned}
requirements &\begin{cases} 0 \le m_c^+ \le 0.112 \\ -0.183 \le m_c^- \le -0.035 \end{cases} \\
design &\begin{cases} 0 \le m_c^+ \le 0.14 \\ -0.208 \le m_c^- \le -0.048 \end{cases} \\
coding &\begin{cases} 0 \le m_c^+ \le 0.140 \\ -0.19 \le m_c^- \le -0.053 \end{cases}
\end{aligned}
\tag{2.9}
$$

The space of tunings for defect removal features are:

$$
\begin{aligned}
\forall x \in \{1..6\} \quad & SF_i = m_d(x-1) \\
requirements: \quad & 0.08 \le m_d \le 0.14 \\
design: \quad & 0.1 \le m_d \le 0.156 \\
coding: \quad & 0.11 \le m_d \le 0.176
\end{aligned}
\tag{2.10}
$$

where m_d denotes the effect of d on *defect removal*.

	rely= very low	rely= low	rely= nominal	rely= high	rely= very high
sced= very low	0	0	0	1	2
sced= low	0	0	0	0	1
sced= nominal	0	0	0	0	0
sced= high	0	0	0	0	0
sced= very high	0	0	0	0	0

Figure 2.10: An example risk table

2.1.4 Threat

The THREAT model returns a heuristic estimate of the threat of a schedule over run in the project. This estimation model is dependent upon the COCOMO effort multipliers

Internally, THREAT contains dozens of tables of the form of 2.10. Each such table adds some "threat" value to the overall project risk when multiplied by the effort multiplier values of the corresponding COCOMO attributes. There are six major categories: schedule, product, personnel, process, platform and reuse. After the threat for each category is calculated, the sum is normalized to produce the final threat rating.

2.10 can be represented as an exponentially decaying function that peaks in one corner of the risk table at a value of two. All the tables peak at either a value of two or four. Since this model is heuristic in nature, the exact height of the peak is not certain. When we perform Monte Carlo simulations over THREAT, we vary the height of the peak by a random factor $0.5 \leq x \leq 1$ if the peak is four, and $0.5 \leq x \leq 1.5$ if the peak is two.

2.2 Local Calibration(LC)

This section describes local calibration (LC), the standard regression procedure used by the COCOMO community. LC is used in conjunction with local data to calibrate the COCOMO II model, as well as the other COCOMO based models.

LC assumes that a matrix $D_{i,j}$ holds:

```
Total_threat =
(Schedule_threat + Product_threat + Personnel_threat + Process_threat + Platform_threat + Reuse_threat)/3.73

Schedule_threat=
Sced_Rely_threat + Sced_Time_threat + Sced_Pvol_threat + Sced_Tool_threat + Sced_Acap_threat + Sced_Aexp_threat +
Sced_Pcap_threat + Sced_Plex_threat + Sced_Ltex_threat + Sced_Pmat_threat

Product_threat =
Rely_Acap_threat + Rely_Pcap_threat + Cplx_Acap_threat + Cplx_Pcap_threat + Cplx_Tool_threat + Rely_Pmat_threat +
Sced_Cplx_threat + Sced_Rely_threat + Sced_Time_threat + Ruse_Aexp_threat + Ruse_Ltex_threat

Personnel_threat =
Pmat_Acap_threat + Stor_Acap_threat + Time_Acap_threat + Tool_Acap_threat + Tool_Pcap_threat + Ruse_Aexp_threat +
Ruse_Ltex_threat + Pmat_Pcap_threat + Stor_Pcap_threat + Time_Pcap_threat + Ltex_Pcap_threat + Pvol_Plex_threat +
Sced_Acap_threat + Sced_Aexp_threat + Sced_Pcap_threat + Sced_Plex_threat + Sced_Ltex_threat + Rely_Acap_threat +
Rely_Pcap_threat + Cplx_Acap_threat + Cplx_Pcap_threat + Team_Aexp_threat

Process_threat =
Tool_Pmat_threat + Time_Tool_threat + Tool_Pmat_threat + Team_Aexp_threat + Team_Sced_threat + Team_Site_threat +
Sced_Tool_threat + Sced_Pmat_threat + Cplx_Tool_threat + Pmat_Acap_threat + Tool_Acap_threat + Tool_Pcap_threat +
Pmat_Pcap_threat

Platform_threat =
Sced_Time_threat + Sced_Pvol_threat + Stor_Acap_threat + Time_Acap_threat + Stor_Pcap_threat + Pvol_Plex_threat +
Time_Tool_threat

Reuse_threat =
Ruse_Aexp_threat + Ruse_Ltex_threat
```

Figure 2.11: THREAT: the calculations.

- The natural log of the *LOC* (lines of code) estimates;
- The natural log of the actual efforts for each project j;
- The natural logarithm of the cost drivers (the scale factors and effort multipliers) at locations $1 \leq i \leq 15$ (for COCOMO 81) or $1 \leq i \leq 22$ (for COCOMO-II).

With those assumptions, Boehm [11] shows that for COCOMO 81, the following calculation yields estimates for "a" and "b" that minimize the sum of the squares of residual errors:

$$\left.\begin{array}{rcl} EAF_i & = & \sum_j^N D_{i,j} \\ a_0 & = & t \\ a_1 & = & \sum_i^t KLOC_i \\ a_2 & = & \sum_i^t (KLOC_i)^2 \\ d_0 & = & \sum_i^t (actual_i - EAF_i) \\ d_1 & = & \sum_i^t ((actual_i - EAF_i) * KLOC_i) \\ b & = & (a_0 d_1 - a_1 d_0)/(a_0 a_2 - a_1^2) \\ a_3 & = & (a_2 d_0 - a_1 d_1)/(a_0 a_2 - a_1^2) \\ a & = & e^{a_3} \end{array}\right\} \quad (2.11)$$

		vl	l	n	h	vh	xh
	rely						
sced	vl				1	2	
	l					1	
	cplx						
sced	vl				1	2	4
	l					1	2
	n						1
	time						
sced	vl				1	2	4
	l					1	2
	n						1
	pvol						
sced	vl				1	2	
	l					1	
	tool						
sced	vl	2	1				
	l	1					
	pexp						
sced	vl	4	2	1			
	l	2	1				
	n	1					
	pcap						
sced	vl	4	2	1			
	l	2	1				
	n	1					
	aexp						
sced	vl	4	2	1			
	l	2	1				
	n	1					
	acap						
sced	vl	4	2	1			
	l	2	1				
	n	1					
	ltex						
sced	vl	2	1				
	l	1					
	pmat						
sced	vl	2	1				
	l	1					

		vl	l	n
	acap			
rely	n	1		
	h	2	1	
	vh	4	2	1
	pcap			
rely	n	1		
	h	2	1	
	vh	4	2	1
	acap			
cplx	h	1		
	vh	2	1	
	xh	4	2	1
	pcap			
cplx	h	1		
	vh	2	1	
	xh	4	2	1
	tool			
cplx	h	1		
	vh	2	1	
	xh	4	2	1
	pmat			
rely	n	1		
	h	2	1	
	vh	4	2	1
	acap			
pmat	vl	2	1	
	l	1		
	acap			
stor	h	1		
	vh	2	1	
	xh	4	2	1
	acap			
time	h	1		
	vh	2	1	
	xh	4	2	1
	acap			
tool	vl	2	1	
	l	1		
	pcap			
tool	vl	2	1	
	l	1		

		vl	l	n
	aexp			
ruse	h	1		
	vh	2	1	
	xh	4	2	1
	ltex			
ruse	h	1		
	vh	2	1	
	xh	4	2	1
	pcap			
pmat	vl	2	1	
	l	1		
	pcap			
stor	h	1		
	vh	2	1	
	xh	4	2	1
	pcap			
time	h	1		
	vh	2	1	
	xh	4	2	1
	pcap			
ltex	vl	4	2	1
	l	2	1	
	n	1		
	pexp			
pvol	h	1		
	vh	2	1	
	pmat			
tool	vl	2	1	
	l	1		
	tool			
time	vh	1		
	xh	2	1	
	aexp			
team	vl	2	1	
	l	1		
	sced			
team	vl	2	1	
	l	1		
	site			
team	vl	2	1	
	l	1		

Figure 2.12: THREAT: the details. For example, looking at the top-left matrix, the `Sced_Rely_risk` is highest when the reliability is very high but the schedule pressure is very tight.

In the case of COCOMO 81 [11] these a, b values are used in the following equation to generate effort estimates. In this equation, EM_i are the effort multipliers from Figure 2.1:

$$effort = a \cdot KSLOC^b \cdot \prod_i^{15} EM_i$$

What is not widely appreciated is the size of the variance in the (a, b) values. The left-hand-side of Figure 2.14 shows the COCOMO-I (a, b) values learned by Baker [6] after, 300 times, extracting

feature	low	high	feature	low	high
aa	1	6	time	3	6
peer	1	6	stor	3	6
ett	1	6	pvol	2	5
prec	1	6	acap	1	5
flex	1	6	pcap	1	5
resl	1	6	pcon	1	5
team	1	6	apex	1	5
pmat	1	6	plex	1	5
rely	1	5	ltex	1	5
cplx	1	6	tool	1	5
data	2	5	sced	1	5
docu	1	5	site	1	6
ruse	2	6	Ksloc	1	980

Figure 2.13: The default ranges for ratings of the COCOMO II attributes according to COCOMO II.2000 [15]

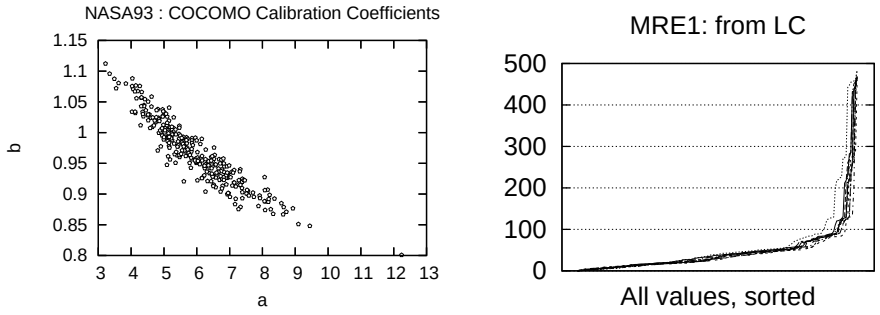


Figure 2.14: Results of applying LC numerous times to 90% of the NASA93 data sets (available from `http://promisedata.org/data`). Left-hand-side shows computed (a,b) values. Right-hand-side shows MREs generated in 20 repeats over the NASA93 data set.

10 projects at random from COCOMO data sets, then applying Equation 2.3 to the remaining data. The data sets used in this study contained 93 projects, so LC was applied to $\frac{93-10}{93} = 89\%$ of the data. A pre-experimental intuition was that we were using enough of the data to yield stable (a,b) values. As can be clearly seen by the wide variance on the (a,b) values in Figure 2.14, this was not the case.

The right-hand-side of Figure 2.14 shows the magnitude of the relative error (or *MRE*) values seen in Baker's study (MRE is a standard measure in the effort estimation field as follows: $MRE = \frac{abs(actual-predicted)}{actual}$).

Some of the MRE errors are very large (up to nearly 500%) suggesting that LC was incomplete

or that the variance in the (a,b) calculations have significant impact on the estimation. In fact, after thirty 90% random samples of that NASA data, Baker found that the a, b ranges were surprisingly large:

$$(2.2 \leq a \leq 9.18) \wedge (0.88 \leq b \leq 1.09)$$

Note that the right-hand-side Figure 2.14 is not without precedent in the estimation literature: it is a well-established result that initial development effort estimates may be incorrect by a factor of four [11] or even more [49].

Elsewhere we have been partially successful in reducing estimation variance of Figure 2.14 using feature subset selection (FSS) [19, 63] or more data collection. Unfortunately, FSS reduces but does not eliminate the a,b variance. Also, further data collection is possible, but only at great organizational expense. This is due to data not being collected or the business sensitivity associated with the data as well as differences in how the metrics are defined, collected and archived.

2.3 Search Based Software Engineering

The contribution of this thesis falls within the area of search based software engineering [21, 39]. This is a diverse area of study, and also a young one, that has attempted to deal with many different phases of the software development life cycle. These vary from software testing to requirements engineering, among others. This thesis, as the title suggests, deals with software process control. Search based software engineering (SBSE) is done by using a combination of a chosen search technique, combined with a fitness function that evaluates the results.

2.3.1 Search Algorithms

There are a variety of meta-heuristic search algorithms that are used in SBSE. Some of these are widely used and considered baseline algorithms, while others are custom made algorithms that

rely on domain knowledge to optimize their performance. Crawford and Baker [26] offer one explanation for the strange success of stochastic search. For models, such as the ones used in this thesis, the solutions are a small part of the total space. A complete search wastes much time exploring uninformative areas of the problem. A stochastic search, on the other hand, does not get stuck in such uninformative areas.

Genetic Algorithms

Genetic algorithms (GA) [43] are AI search algorithms that attempt to immitate the population based evolutionary model of nature. The algorithm initially starts with a random population of solutions. These solutions, individually referred to as chromosomes, are iteratively seleted, mutated, and recombined based on how well they score with a predefined fitness function. This eventually increases the fitness of the population, after which the algorithm stops when reaching a certain performance milstone in a chromosome, or when it has gone through a certain number of generations. GA has been historically widely used in the field of SBSE, and is very much considered and standard SBSE algorithm.

Reactive Tabu Search

Tabu search [33] is a local search algorithm that allows the jump to sub-optimal non-improving solutions in order to avoid local optima. Tabu search uses a list, called the *tabu list*, that keeps a record of the recent search history of the algorithm. This is done in order to avoid visiting the same solution repeatedly so that when a sub-optimal jump does occur, a new solution in the search space is being explored. Note that old solutions are not banned forever, but rather they are restricted for a while. This algorithm was developed in 1986 as a response to the development of SA, and like it is based on an analogy of a natural phenomenon.

Simulated Annealing

Simulated Annealing is the AI search algorithm that is used in this work. It belongs to the family of Monte Carlo algorithms. *Monte Carlo* algorithms randomly sample the space of possible controllable model states. A *Metropolis* Monte Carlo algorithm [65] creates new states by small mutations to some *current* state. If a new state is "better" (as assessed via an *energy function*), it becomes the new *current* state used for future mutations. Otherwise, a Boltzmann acceptance criteria is used to probabilistically decide to assess the new state: the worse the new state, the less likely that it becomes the new current state. The algorithm is silent on the mutation mechanism. For our experiments, we freeze $\frac{2}{3}$ of the features and randomly select ranges for the rest.

In 1983, Kirkpatrick et al. [51] proposed a modification that was inspired by a procedure used to make the strongest possible glass. Initially, glass is heated to allow atoms to move freely. The temperature is then slowly lowered such that the atoms can find the stablest configuration with lowest energy. A *simulated annealing* (SA) algorithm adds a "temperature" variable to the Boltzmann accept criteria such that, at high temperatures, it is more likely that the algorithm will jump to a new worse current state. This allows the algorithm to jump out of local minima while sampling the space of options, a useful ability. As the temperature cools, such jumps become less likely and the algorithm reverts to a simple hill climber.

The acceptance criteria P for a new state is defined using the current temperature T, the energy of the current solution (e), and the energy of the new mutation (en):

$$P(e, en, T) = e^{(e-en)/T} \tag{2.12}$$

T is defined to *decrease* as the simulator loops through $k = 1 \ldots kmax$ iterations. We use $T = e^{-100*k/kmax}$.

Two advantages of SA algorithms are their *implementation simplicity* and their *ability to handle*

```
function sa(kmax)
  s := s0; e := E(s)           // Initial state, energy.
  sb := s; eb := e             // Initial "best" solution
  k := 0                       // Energy evaluation count.
  while k < kmax and e > emax  // Loop
    sn := neighbour(s)         //   Pick some neighbour.
    en := E(sn)                //   Compute its energy.
    if en < eb then            //   Is this a new best?
      sb := sn; eb := en       //     Yes, save it.
    if random() < P(e, en, temp(k/kmax))
     then s := sn; e := en     //     Maybe jump
    k := k + 1                 //   One more evaluation done
  return sb                    // Return best
```

Figure 2.15: SA pseudo-code: a new solution *sn* (with new energy *en*) replaces the current solution if (a) it has a lower energy or (b) the acceptance predicate *P* endorses it. Only in the case of (a) should the new solution replaces the current best solution.

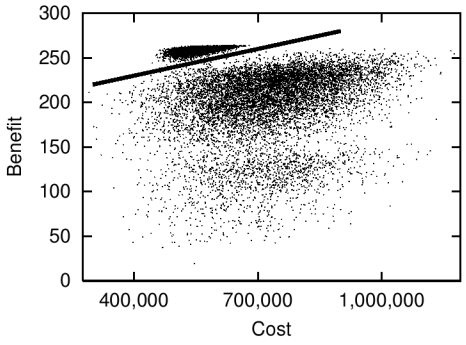


Figure 2.16: Processing a JPL requirements model.

non-linear models:

- *Implementation simplicity:* 2.15 illustrates the simplicity of the algorithm. Memory is only required for one current solution (*s*), one new solution (*sn*) and one best solution (*sb*) that stores the best solution seen at any time in the simulation.

- *Non-linear models:* Previously [24, 32], SA was applied to non-linear JPL requirements models where minimizing the cost of project mitigation can decrease the number of requirements achieved by that project. Hence, decreasing *both* the cost and achieved requirements is a non-linear problem that must trade between minimizing cost and increasing requirements

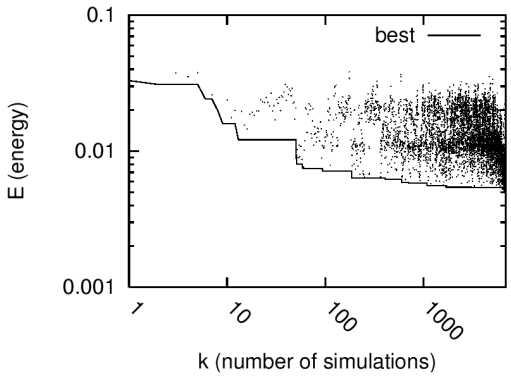


Figure 2.17: Dots & lines are SA output from current & best solution (respectively) after k simulations.

coverage. The top-left line of Figure 2.16 divides the behavior of the JPL requirements models before and after simulated annealing. As shown below the line, initial Monte Carlo sampling of the possible mitigation lead to a large range of costs and benefits. Simulated annealing found a set of mitigation that lead to the small cloud of solutions above the line. Compared to the initial samples, these new solutions had decreased cost, increased benefit (number of requirements covered), and decreased variance (shrank the space of solutions).

Two disadvantages of SA algorithms are their *incompleteness* and the *complexity* of their solutions:

- *Incompleteness:* In our domain, we have some evidence that the incomplete nature of the heuristic SA search is not a major problem. 2.17 shows a sample run of our SA tool running on our prediction models for $K = 10,000$ simulations. As k increases for $1 \ldots 10,000$, it becomes less and less likely that a better *best* has been missed. Hence, we run our simulations for ten times the period it takes for *best* to stabilize (at $k \approx 1000$).

- *Solution complexity:* Simulated annealing offers constraints to all controllable features. Often this is an over-constrained solution since, in many domains, a repeated empirical result is a *feature subset selection* effect; i.e. models that constrain M variables perform just as well, or better, than models that constrain N variables ($M \ll N$) [37,53]. For example, Kohavi [53]

	av. number of features			
dataset	before	after	$\frac{after}{before}$%	accuracy change
breast cancer	10	2.9	29%	+0.14%
cleve	13	2.6	2%	+5.89%
crx	15	2.9	19%	+4.49%
DNA	180	11	6%	+3.63%
horse-colic	22	2.8	13%	+1.63%
Pima	8	1	13%	+0.79%
sick-euthyroid	25	4	16%	+0.38%
soybean	35	12.7	36%	+0.15%
average	38.5	4.99	19%	+2.14%

Figure 2.18: Feature subset selection results from [53]

studied some machine learners to find that using just 19% of the available features *increased* prediction accuracy by just 2.14% (on average). For another example, when feature subset selection was applied to the JPL requirements model of Figure 2.16, we found that up to $\frac{2}{3}$-rds of the features can be left unconstrained, without effecting the conclusions [32].

Other Algorithms

Harman [40] discusses several standard search algorithms that are used in the field of SBSE, such as simulated annealing and genetic algorithms. However, SBSE need not be restricted to only these standard algorithms, as there are other custom made AI search algorithms that attempt to perform the same functions as these standard algorithms. We will present two such algorithms here.

KEYS Keys is a custom AI serach engine that was developed and compared to other standard algorithms in [35].

The premise of KEYS is that within the space of possible decisions, there exist a small number of *key* decisions that determine all others [57].

If a model contains keys, then a general search through a large space of options is superfluous. A better (faster, simpler) approach would be to just explore the keys. KEYS uses support-based Bayesian sampling to quickly find these important variables.

There are two main components to KEYS - a greedy search and the BORE ranking heuristic. This is in some ways similar to part of the STAR Algorithm general structure, which is further explained in §3.3.

The greedy search explores a space of M mitigation over the course of M "eras". Initially, the entire set of mitigations is set randomly. During each era, one more mitigation is set to $M_i = X_j$, $X_j \in \{true, false\}$.

In Keys, each era e generates a set $< input, score >$ as follows:

1: $MaxTries$ times repeat:

- $Selected[1 \ldots (e-1)]$ are settings from previous eras.
- *Guessed* are randomly selected values for an unfixed mitigation.
- $Input = selected \cup guessed$.
- Call the fitness function to compute the *score*.

2: The $MaxTries$ *score*s are divided into β% "best" while the remainder are sent to "rest".

3: The mitigation values in the *input* sets are then scored using BORE (described below).

4: The top ranked mitigation (the default is one, but the user may fix multiple mitigation at once) are fixed and stored in $selected[e]$.

The search moves to era $e+1$ and repeats steps 1,2,3,4. This process stops when every mitigation has a setting. For full details, see Figure 2.19.

KEYS ranks mitigation by combining a novel support-based Bayesian ranking measure, which is detailed in §3.2.

SEESAW SEESAW is a novel stochastic stability tool that (a) considers a very large set of minor changes using stochastic sampling; and (b) carefully selects the right combination of effective

```
Procedure KEYS
while FIXED_MITIGATIONS != TOTAL_MITIGATIONS
   for I:=1 to 100
      SELECTED[1...(I-1)] = best decisions up to this step
      GUESSED = random settings to the remaining mitigations
      INPUT = SELECTED + GUESSED
      SCORES= SCORE(INPUT)
   end for
   for J:=1 to NUM_MITIGATIONS_TO_SET
     TOP_MITIGATION = BORE(SCORES)
     SELECTED[FIXED_MITIGATIONS++] = TOP_MITIGATION
  end for
end while
return SELECTED
```

Figure 2.19: Pseudo-code for KEYS

minor changes. Figure 2.20 shows SEESAW's pseudo-code. The code is an adaption of Kautz & Selman's MaxWalkSat local search procedure [48]. The main changes are that each solution is scored via a Monte Carlo procedure (see `score` in Figure 2.20) and that SEESAW seeks to *minimize* that score.

SEESAW was designed based on observing the properties of USC software engineering models §2.1. Based on that domain knowledge, the most interesting ranges in the features was found to be the minimum and maximum values. This observation was due to results produced by STAR (explained in §3.3) in such papers as [60]. The reason for this is simple: All the functions in the above indicated models are monotonic, causing the most dramatic effects to occur at the extreme ends of the ranges. In fact, SEESAW takes its name from the way the algorithm seesaws between extreme values.

SEESAW is a stochastic algorithm: the selection of the next feature to explore is completely random. SEESAW incrementally grows solutions from unconstrained (where all features can take any value in $\{Low, High\}$) to fully constrained (where all features are set to a single value). This is unlike simulated annealing or MaxWalkSat, which simultaneously offer settings to all features at every step of their reasoning. Figure 2.21 shows a single run of SEESAW: each dot marks one selection (lines 16,17,18 of Figure 2.20). For example, if Figure 3.1 includes a process maturity

```
function run (AllRanges, ProjectConstraints) {
  OutScore = -1
  P = 0.95
  Out = combine(AllRanges, ProjectConstraints)
  Options = all Out features with ranges low < high
  while Options {
    X = any member of Options, picked at random
    {Low, High} = low, high ranges of X
    LowScore = score(X, Low)
    HighScore = score(X, High)
    if LowScore < HighScore
      then Maybe = Low;  MaybeScore = LowScore
      else Maybe = High; MaybeScore = HighScore
    fi
    if MaybeScore < OutScore or P < rand()
      then delete all ranges of X except Maybe from Out
      delete X from Options
      OutScore = MaybeScore
    fi
  }
 return backSelect(Out)
}
function score(X, Value) {
  Temp = copy(Out)  ;; don't mess up the Out global
  from Temp, remove all ranges of X except Value
  run monte carlo on Temp for 100 simulations
  return median score from monte carlo simulations
}
```

Figure 2.20: Pseudocode for SEESAW.

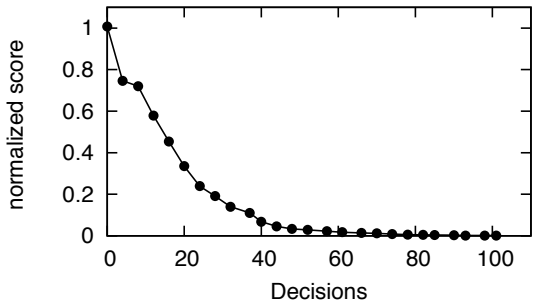


Figure 2.21: Single run of SEESAW, score normalized min..max to 0..1

of (3,4,5) and SEESAW selects "5", then that is three decisions. (this is why the 29 dots of Figure 2.21 result in 100 decisions). Note how, as decisions are made, the score is minimized. Score minimization is desirable since the scores are calculated from a combination of project predictions that we want to reduce (total effort, defects, development time) based on the USC models.

XOMO and TAR3 XOMO [62] is the work that preceded STAR. It was designed as a framework for Monte-Carlo simulations made to run on USC software related models §2.1. The search technique that is used by this framework is a data miner called TAR3 (and evolution of TAR2 [61]). TAR3 can be described as a treatment learner, returning concise rules or "treatments" that have the largest effect on the output of the fitness function used. Having small rules is useful for software process control since this reduces the managerial interventions on a software project, while at the same time ensuring that such interventions will have high utility for the project. Note that XOMO is the first tool here that integrates multiple models, these being COCOMO, COQUALMO and the threat model. All previously mentioned tools operate on a single model, that being either COCOMO or another project specific model.

TAR3 inputs a set of training examples E. Each example maps a set of attribute ranges to some class symbol; i.e. $\{R_i, R_j, ... \rightarrow C\}$ The class symbols $C_1, C_2..$ are stamped with some utility score that ranks the classes; i.e. $\{U_1 < U_2 < .. < U_C\}$. With E, these classes occur at frequencies $F_1\%, F_2\%, ..., F_C\%$. A "treatment" T of size X is a conjunction of attribute ranges $\{R_1 \wedge R_2 ... \wedge$

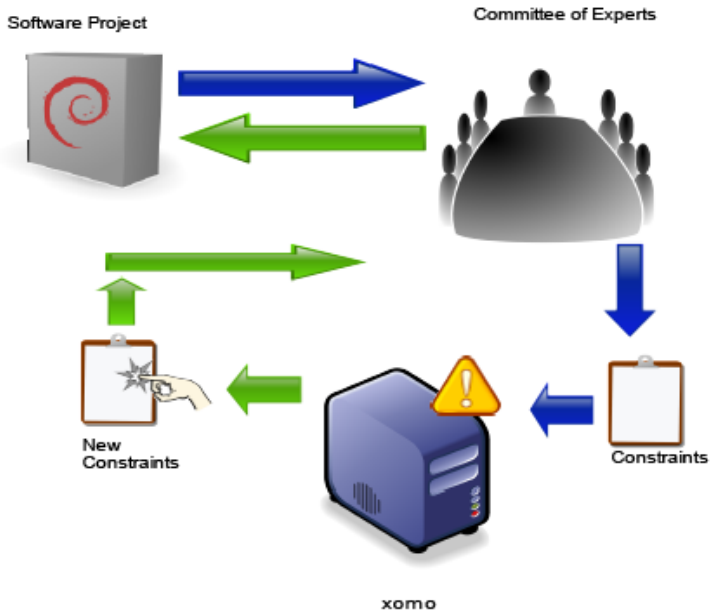


Figure 2.22: A diagram offering an overview of where and how XOMO is meant to operate. This provided the inspiration to make STAR.

$R_X\}$. Some subset of $e \subseteq E$ are consistent with the treatment. In that subset, the classes occur at frequencies $f_1\%, f_2\%, ...f_C\%$. TAR3 seeks the seek smallest T which most changes the weighted sum of the utilities times frequencies of the classes. Formally, this is called the $lift$ of a treatment:

$$lift = \frac{\sum_C U_C f_C}{\sum_C U_C F_C}$$

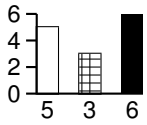
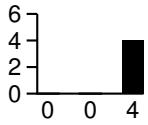
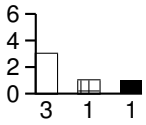
For example, consider the log of golf playing behavior seen in 2.23. In that log, we only play *lots* of golf in $\frac{6}{5+3+6} = 43\%$ of cases. To improve our game, we might search for conditions that increases our golfing frequency. Two such conditions are shown in the `WHERE` test of the select statements in Figure 2.24. In the case of `outlook= overcast`, we play *lots* of golf all the time. In the case of `humidity` $\leq$ `90`, we only play *lots* of golf in 20% of cases. So one way to play lots of golf would be to select a vacation location where it was always overcast. While on holidays, one thing to watch for is the humidity: if it rises over 90%, then our frequent golf games are threatened.

outlook	*temp(^{o}F)*	*humidity*	*windy?*	*class*
sunny	*85*	*86*	*false*	*none*
sunny	*80*	*90*	*true*	*none*
sunny	*72*	*95*	*false*	*none*
rain	*65*	*70*	*true*	*none*
rain	*71*	*96*	*true*	*none*
rain	*70*	*96*	*false*	*some*
rain	*68*	*80*	*false*	*some*
rain	*75*	*80*	*false*	*some*
sunny	*69*	*70*	*false*	*lots*
sunny	*75*	*70*	*true*	*lots*
overcast	*83*	*88*	*false*	*lots*
overcast	*64*	*65*	*true*	*lots*
overcast	*72*	*90*	*true*	*lots*
overcast	*81*	*75*	*false*	*lots*

Figure 2.23: TAR3: Playing golf.

input:	`SELECT class`
	`FROM golf`
		`SELECT class`
		`FROM golf`
		`WHERE`
		`outlook = 'overcast'`
			`SELECT class`
			`FROM golf`
			`WHERE`
			`humidity >= 90`

output:	`none none none none none`
	`some some some lots lots`
	`lots lots lots lots`
		`lots lots lots lots`	`none none none some lots`

distributions:			

legend: none some lots

Figure 2.24: TAR3: Class distributions selected by different conditions in Figure 2.23.

The tests in the `WHERE` clause of the select statements in Figure 2.24 is a treatment. Classes in treatment learning get a score U_C and the learner uses this to assess the class frequencies resulting

from *applying a treatment* (i.e. using them in a `WHERE` clause). In normal operation, a treatment learner does *controller learning* that finds a treatment which selects for better classes and reject worse classes By reversing the scoring function, treatment learning can also select for the worse classes and reject the better classes. This mode is called *monitor learning* since it finds the thing we should most watch for. In the golf example, *outlook* = *'overcast'* was the controller and $humidity \geq 90$ was the monitor.

Formally, treatment learning is a weighted-class minimal contrast-set association rule learner. The treatments are associations that occur with preferred classes. These treatments serve to contrast undesirable situations with desirable situation where more of the outcomes are favorable. Treatment learning is different to other contrast set learners like STUCCO [9] since those other learners don't focus on minimal theories.

Conceptually, a treatment learner explores all possible subsets of the attribute ranges looking for good treatments. Such a search is impractical in practice so the art of treatment learning is quickly pruning unpromising attribute ranges. This study uses the TAR3 treatment learner [45] that uses stochastic search to find its treatments.

Sometimes, one round of TAR3 is not enough. *Iterative TAR3* runs by conducting multiple Monte Carlo simulations over the ranges of any uncertain variables. For example, there are 28 variables in XOMO, which uses the USC models:

- Ksloc;
- 5 scale factors;
- 17 effort multipliers;
- 2 calibration parameters ("`a,b`");
- 3 defect removal activities (automated analysis, peer reviews, execution testing and tools).

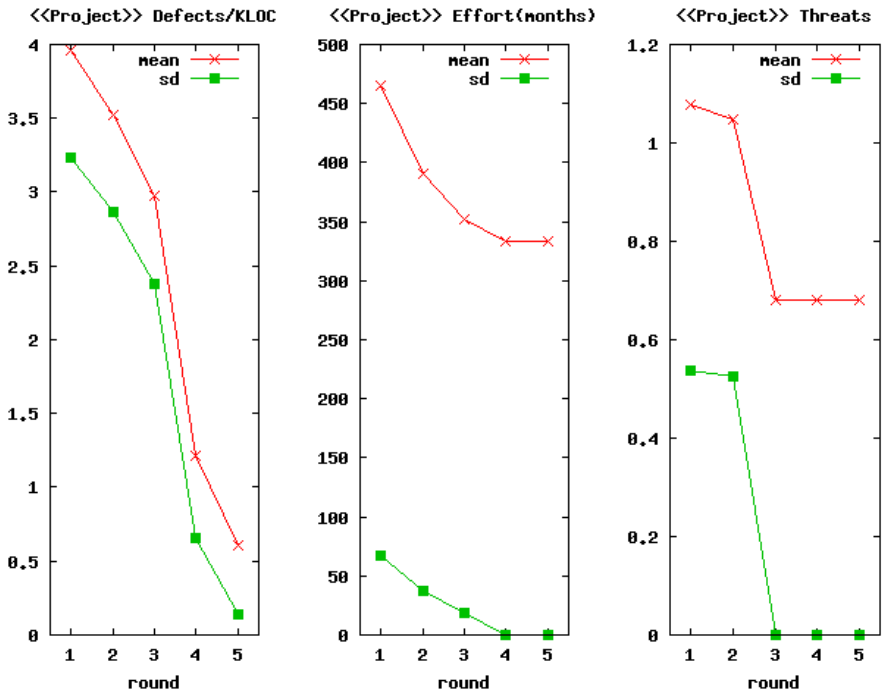


Figure 2.25: The reductions that XOMO was able to produce in the model estimates.

In the case studies that follow, only a partial description of some of these variables are available. Hence, learning is a process of sampling from the known constraints, finding the best treatment, then revising the constraints. After, say, 1000 Monte Carlo runs, BORE(§3.2), short for "best or rest", classifies the outputs as (say) either the 333 *best* or 667 *rest*. The treatment learner studies the results and notes which input ranges select for *best*. The ranges found by the learner then become restraints for future simulations.

Using XOMO and TAR3, we were able to able to produce stable conclusions within the space of options provided to us in a project, as well as achieve control over the estimates of the models. Figure 2.26 demonstrates one instance of such stability, and shows that some attributes were chosen regularly for the specific project used. Figure 2.25, on the other hand, shows that at the same time XOMO was able to offer control over the estimates of a project, reducing the mean and the standard deviation of those estimates. Further discussing these results is outside the scope of this thesis.

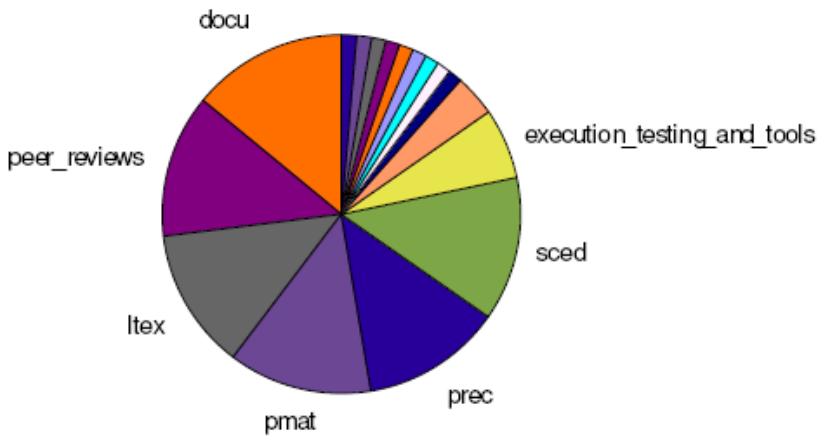


Figure 2.26: The occurance of attributes in the TAR3 recommendations over 10 runs of XOMO.

2.3.2 Applications of SBSE

SBSE is being studied for use in tackling many software engineering issues, some of which will be mentioned here and briefly described. This section is presented in the form of a brief literature survey, and aims to present the reader with the diversity of the field of SBSE.

Project Planning and Cost Estimation

In [2], software project simulators, such as the likes of STELLA, VENSIM, ITHINK and POSER-SIM, are used to generate a search space of possible decisions along with the outcome of using such decisions. This search space is passed to an evolutionary algorithm similar in concept to a genetic algorithm. By evaluating decisions using a fitness function based on how well the rule classifies project examples, this algorithm is able to generate a hierarchy of decision rules. They are able to demonstrate how applying these rules improves the management of the software project. Note that the model used for this system is project specific rather than an industry standard software engineering model.

In [4], a scheme is developed for project management and allocating work loads to teams. The goal of this scheme is to determine the optimal flow of "work packages" (WP) and at the

same time determine the optimal team organization given a predetermined staff allocation. This being an NP-Hard task, the authors elect to use search based techniques, genetic algorithms (GA) specifically. Note that each of those issues has a dedicated GA for solving it, and that the fitness function is based on queuing theory [36]. Their system alternates back and forth between the two problems till equilibrium is achieved, after which the final output of development time is compared to a supplied deadline. If the deadline isn't met, rethinking staffing is recommended. In addition to this, simulation for different project scenarios (involving reworking and/or abandoning parts of the project) is also undertaken. The estimation model is not specified. An empirical study was conducted on a massive Y2K maintenance intervention conducted on a large financial software system from a European financial organization. Using this study they were able to demonstrate the ability of their scheme of being able to find an optimal or near-optimal solution, and hence reduce development time. A followup study was conducted [3] where hill climbing and simulated annealing were also used, and the results from all three algorithms were compared. GA was found to more efficient in finding the best solution; however, is was also found that the simpler methods (hill climbing, simulated annealing) also eventually caught up to GA given additional iterations.

In [52], case-based reasoning is used for predicting project effort. The method used for this, "Angel", was used on the "Finnish" data set. This data set is comprised of 407 cases, each described by 90 features. It should be noted that these features aren't COCOMO features, rather they are specific to that particular dataset. It should also be noted that the COCOMO model, and standard models in general, are criticized as being too specific to the data set that they are derived from. In this study, feature subset selection is used in order to determine the most significant features to use in predicting the effort. In order to do that, three search based methods were used: Random Search (RS), Hill Climbing (HC), and Forward Sequential Selection (FSS). While they all reduce the error rates compared to using all the features, HC and FSS produced the best results, with FSS edging out HC due to its quicker performance and more stable results.

Software testing

In [8], the authors are using evolutionary testing that's based on genetic algorithms to generate test data for software projects. The main point of the paper is exploring a testability transformation [41] algorithm that allows the evolutionary testing system used to handle code with loop-assigned flags. The system used was the Daimler Chrysler Evolutionary Testing System [47] running on top of the Matlab Genetic and Evolutionary toolbox [72]. The empirical studies were conducted on several code bases, one of which was a the code for a car navigation system.

In [18], a genetic algorithm based system is utilized in order to determine test cases where a given real-time system is apt to failure, effectively stress testing such systems. Failure here is defined as not being able to complete a time critical task within the time limits set for that task to be completed. The testing system prototype is called the Real-Time Test Tool (RTTT), and is based on using GAlib, a C++ genetic algorithm framework, and a POSIX compliant scheduler that emulates single processor execution. RTTT was used in a case study on a generic avionics platform that was a joint effort between the software engineering institute , the Naval weapons center and IBM's federal sector division. In this scenario and others, RTTT was found to be able to automatically generate test cases that were able to stress real-time systems. Note that a single GA execution run under this system took a mean time of 46.5 minutes.

In [54], regression testing and test case prioritization are the tasks being tackled. Test case prioritization is conducted due to the lack of enough resources to run all the regression test cases available for a project, and in order to improve the effectiveness of the regression tests being used. The main purpose of the paper is to compare several algorithms in there ability to generate prioritized test cases. The algorithms compared were greedy, additional-greedy, 2-optimal, hill climbing, and genetic algorithms. The first three are greedy algorithms that might settle for a local minimum solution. Three fitness functions were utilized and involved a percent measure of coverage of certain code aspects. After conducting empirical studies on code bases of varying sizes and with varying availability of test suites, it was found that plain greedy search performed

the worst, while all the other algorithms performed very closely to each other.

Automated maintenance

The authors of [66] use a tool called "Bunch" in order to generate abstract high level representations of software. It does so by clustering module dependency graphs (MDG) using a search algorithm that is integrated within Bunch to examine a small subset of all possible clustering outcomes. The algorithm used within Bunch is hill climbing, with the option of using simulated annealing to avoid getting stuck in local optima. Note that the MDG's are produced by other language specific tools, making Bunch language independent. However, due to the nature of the algorithms used, the clustering results aren't always identical, and it isn't always clear why MDG's are clustered the way they are, sometimes requiring user intervention to clear things up a bit. Case studies are demonstrated on the dot graphing tool, the korn shell and a proprietary AT&T file system, showing how it is possible using the tool to generate a high level abstraction of the software. This is ultimately meant to facilitate software maintenance on software engineers by giving them an overview of the software that is simpler to understand.

Automated refactoring of code is explored in [70]. This refactoring is aimed at simplifying software maintenance by increasing flexibility, reusability and understandability according to the Object-Oriented Design Quality Assessment (QMOOD) [7]. A system called *CODe-Imp* was developed which uses the Java program model (JPM) to assess legal refactoring that is to be applied to the abstract syntax tree (AST) of java 1.4 based programs. Possible types of refactoring include moving fields and methods between the classes according to the hierarchy, and modifying the class hierarchy by inserting or removing intermediate classes. There were three search methods used by this system: First-Ascent Hill Climbing (HC1), Steepest-Ascent Hill climbing (HC2), and Simulated Annealing (SA). In empirical testing, it was found that all three search methods generated quality improvements, with slightly different results. HC1 was the least computationally intensive, HC2had the most consistent results between runs, and SA producing the greatest improvement in

some cases. In addition, use of the reusability fitness function produced unsatisfactory results.

Other Applications

Several other applications of SBSE exist, amog those are the following:

- **Requirements Engineering:** The Multi-objective next release problem (MONRP) is tackled for the first time in [81]. This is an extension from the next release problem [5], where instead of having a single release requirement or several requirements that are considered independent, there are multiple requirements that might conflict with each other. This is an NP-hard problem with a set of possible solutions that are along a Pareto front. This front needs to be defined so that a decision maker is able to pick an acceptable approximate Pareto optimal solution. The search algorithms used in this study were Random Search, used as the baseline sanity check, and three other algorithms based on genetic algorithms (GA). These were Single-Objective GA, Pareto GA, and Non-dominated Sorting GA II (NSGA-II) [28]. In the empirical study, all the GA's performed better than Random Search, and within the smaller problem instances showed little difference between there results. However, in the larger problem instances that have more requirements, NSGA-II outperformed the others in being able to trace the Pareto front, and further widened the gap between it and Random Search, while Single-Objective GA was better able to find the extreme ranges of the Pareto front. So, it was demonstrated that solving MONRP is feasible.

- **Compiler Optimization:** Cohen et al. [23] attempts to improve garbage collection for multi-threaded applications by clustering threads and creating sub-heaps for each cluster of threads, while maintaining a shared heap among all the thread clusters. The premise is that this will aid in parallelizing garbage collection, while at the same time allowing this method to scale better than providing a heap for every individual thread for very highly threaded programs. The clustering system used is based on hill climbing, and uses thread dependency graphs to

cluster the threads together. The empirical study was conducted on a peer-to-peer file sharing distributed application based on the .Net platform. By clustering the threads, the size of the shared heap was reduced by up to 30%, thus contributing to parallelizing garbage collection and increasing its efficiency.

- **Quality Assessment:** In [17], an approach is proposed to enhance predictive accuracy of software quality by reusing expertise from a several experts, and combining them to form an expert with the optimal predictive accuracy. This approach is modeled as an optimization problem in a large solution space, making it suitable for meta-heuristic search algorithms such as simulated annealing (SA) and genetic algorithms (GA). Both of those algorithms are used in the studies conducted in this paper, with empirical studies conducted on the JAVA API classes. Note that the specific aspect of software quality that is explored is object oriented software stability, where the quality metrics used to define this are dependent on the attributes of the software code base. The experts are modeled as Bayes Classifiers. Each expert is broken up into "expertise chunks" by limiting the input space. These chunks are then combined and modified to increase the final predictive accuracy. The search algorithm is responsible for choosing the chunks, where the fitness function used is the predictive accuracy. The resulting combined expert was shown to be better than any individual expert in terms of predictive accuracy. This showed that even bad experts have good expertise to contribute, where a correlation was shown between the accuracy and the amount of experts used to form the final expert. It was also shown through this study that SA performed comparably to GA, while being slightly quicker.

2.4 Moving on

The reader may wonder why we use a stochastic methods to explore project options. Would not a simpler method suffice? For example, in the case of linear models that have been precisely

26 inputs							*3 outputs*		
rely	*plex*	*ksloc*	...	*pcap*	*time*	*aa*	*effort*	*schedule risk*	*defects*
5	*1*	*118.80*	...	*5*	*3*	*5*	*2083*	*69*	*0.50*
5	*1*	*105.51*	...	*1*	*3*	*5*	*4441*	*326*	*0.86*
5	*4*	*89.26*	...	*3*	*5*	*3*	*1242*	*63*	*0.96*
5	*2*	*89.66*	...	*1*	*4*	*5*	*2118*	*133*	*2.30*
5	*1*	*105.45*	...	*2*	*4*	*5*	*6362*	*170*	*2.66*
5	*3*	*118.43*	...	*2*	*6*	*2*	*7813*	*112*	*4.85*
5	*4*	*110.84*	...	*4*	*4*	*4*	*4449*	*112*	*6.81*
...									

Figure 2.27: XOMO output from [62]. Looking at the first and third lines of results, notice how even though changing project parameters manages to reduce the effort, defects almost doubles.

tuned using local data, it is a simple matter to check if a combination of internal changes improve the project estimates. Many of the relationships inside the COCOMO model, as well as its sister models, are linear. For such models, "what- if" queries require just a *simple linear extrapolation* to assess the relative effectiveness of some combination of internal changes.

Unfortunately, not all tunings are precise. Sometimes, even after tuning, the gradient of the relationships may not be known with certainty. One example is the COCOMO effort model predictions being affected linearly and exponentially by two features a, b. We showed above in §2.2 the Baker result [6] that shows that even with tuning data, variance of the parameters, and the MRE of the resulting estimates, can be very large.

Another drawback with simplistic linear extrapolation is that, when optimizing for effort and time and defects and threats, there may be contradictory effects. For example, Figure 2.27 shown below shows several results, produced by XOMO in subsubsection 2.3.1, where the change in effort doesn't transfer to the same amount of change in defects. Hence, optimizing our models is not a simple matter of moving fixed distances over some linear effect; there are also some trade-offs to be considered.

Figure 2.27 demonstrates how managing the estimates in one of the models does not necessarily translate to the estimates of all the other models. One instance that demonstrates this is comparing

the last and the second row of Figure 2.27, where plex is reduced from *high*(4) to *very low*(1). In this instance defects is vastly reduced, threat increases drastically, and effort maintains its levels. Stochastic, AI search, algorithms are able to solve for such non-linear problems and escape from local minima/maxima, while also finding solutions faster than complete search, and for larger problems [67].

Since we failed to generate precise tunings that yield exact estimates, and given the need to consider combinations of effects, we considered a change in our research goals. In our attempt to find stable conclusions within the space of options provided to us, we created XOMO and were able to achieve that as presented briefly in subsubsection 2.3.1. However, XOMO proved to be too slow, taking as much as 30 minutes to rank features [58]. In addition, it was very hard to maintain and customize the tool, and we were unable to explore the internal variance of the model tunings using XOMO. This drove us to attempt to embrace this variance within the models and use it to our advantage [58, 60] by developing and using another faster, more maintainable and more customizable tool that uses an AI search algorithm to attempt to find stable conclusions. We called this tool STAR.

Chapter 3

STAR and Internal Studies

3.1 The Fitness Function and Heuristic Modifications

When evaluating possible solutions with SA, better solutions have lower score, referred to here as "energy". The fitness function defines the energy E such that $E \geq 0$ and *lower* values are *better*. This fitness function is here defined as the distance to the origin of the 4-D space composed of effort (Ef), defects (De), threats (Th), and development time (Mo). For our purposes, we model :

$$RD = relydefect^{(RELY-3)} \tag{3.1}$$

$$E = \frac{\left(\sqrt{(\overline{Ef}*\alpha)^2+(\overline{De}*(\beta+RD))^2+(\overline{Th}*\gamma)^2+(\overline{Mo}*\delta)^2}\right)}{\sqrt{\alpha^2+(\beta+RD)^2+\gamma^2+\delta^2}} \tag{3.2}$$

Here, $\bar{x}$ is a normalized value $0 \leq \frac{x-min(x)}{max(x)-min(x)} \leq 1$. Hence, our energy ranges $0 \leq E \leq 1$ and *lower* energies are *better*. α, β, γ, and δ are the weighting factors for the different models, all set to 1 by default. $relydefect$ is a constant that defines a relationship between RELY and the defect model: as RELY increases, the weighting of the defect model increases as well. The default

setting for $relydefect$ is 1.8. Initially, this seems like a magic number; however this setting was intentionally chosen such that, as *rely* varies from its minimum to maximum ratings of *very low*(1) to *very high*(5), the *RD* weight factor changes by about a factor of ten as is demonstrated below.

$$RD(rely = 1) = 1.8^{1-3} = 0.31$$
$$RD(rely = 5) = 1.8^{5-3} = 3.24$$

In addition to the above, we implemented additional heuristic modifications to the models relating to the defect reduction measures in the COQUALMO model. These modifications where implemented in the form of loop-backs, which modify the *rely* and the *tool* model attributes in the following manner:

$$\begin{aligned} rely = rely &+ (aa-3>0?aa-3:0) \\ &+ (ett-3>0?ett-3:0) \\ &+ (peer-3>0?peer-3:0) \end{aligned} \tag{3.3}$$

$$\begin{aligned} tool = tool &+ (aa-3>0?aa-3:0) \\ &+ (ett-3>0?ett-3:0) \end{aligned} \tag{3.4}$$

In Equation 3.3 and Equation 3.4, *aa*, *ett* and *peer* are automated analysis, execution-based testing, and peer reviews as indicated by Figure 2.1. The equations are written in C style for simplification. For example, $(aa-3>0?aa-3:0)$ is equivalent to the following:

```
if (aa > nominal)
     return aa-3;
else return 0;
```

These loop-backs attempt to model a logical relationship that exists between the defect reduction measures and the reliability and development and testing tools used in the development of the

software project. The general reasoning is that, as the rating of defect reduction measures increase, this automatically means that *rely* and *tool* are more highly rated. As a counter example, it doesn't make sense for *tool* to be rated at *very low* while at the same time *aa* and *ett* are rated at *very high*, since this inherently indicates that development and testing tools are being heavily used by a project.

The following is an example of how this would work. Assume that *tool* is set at low (2), and *aa* and *ett* at set at high (4). This will modify the rating of *tool* in the following manner: $tool = rating(tool) + [rating(aa) - rating(nominal)] + [rating(ett) - rating(nominal)] = 2 + [4 - 3] + [4 - 3] = 4$, i.e. *tool* would be set to high. Note that for the above two attributes, the values are restricted in accordance to the default model limits shown in Figure 2.13 as well as the limits set by the projects. This means that *rely* and *tool* can never exceed 5, which corresponds to a rating of *very high*.

With the above fitness function applied to the search stage (i.e. the Simulated Annealing stage) in STAR, this stage constrains all the features. For this reason feature subset selection is very important to explore and trim non-essential features in order to produce succinct policies. Before we can *remove* non-essential features, we must first *rank* them according to their effectiveness.

3.2 Ranking Method: Support-Based Bayesian Ranking with "BORE"

BORE [22] (short for "best or rest") divides numeric scores seen over K runs and stores the top 10% in *best* and the remaining 90% scores in the set *rest* (the *best* set is computed by studying the delta of each score to the best score seen in any era). It then computes the probability that a value is found in *best* using Bayes theorem. The theorem uses evidence E and a prior probability $P(H)$ for hypothesis $H \in \{best, rest\}$, to calculate a posteriori probability $P(H|E) = P(E|H)P(H) \ / \ P(E)$.

This ranking method is the default ranking method used in STAR, and is hence used in all the

subsequent experiments that were conducted. STAR ranks the feature ranges seen in K runs of a simulated annealer by dividing the K runs into:

- *Best:* those associated with the BEST% solutions (i.e. those with the BEST% *least* energy);
- And the *rest* (i.e. the other 100-BEST% of solutions).

It then computes the probability that a range is found in *best* using Bayes' Theorem. Informally, the theorem says $next = old * new$ i.e. what we'll believe *next* comes from how *new* evidence effects *old* beliefs. More formally:

$$P(H|E) = P(E|H)P(H) \;/\; P(E) \tag{3.5}$$

i.e. using evidence E and a prior probability $P(H)$ for hypothesis $H \in \{best, rest\}$. The theorem calculates a posteriori probability $P(H|E)$. Simple Bayes classifiers are often called "naive" since they assume independence of each feature. While this assumption simplifies the implementation (frequency counts are required only for each feature), it is possible that correlated events are missed by this "naive" approach. Domingos and Pazzini show theoretically that the independence assumption is a problem in a vanishingly small percent of cases [31]. This explains the repeated empirical result that, on average, seemingly naive Bayes classifiers perform as well as other seemingly more sophisticated schemes (e.g. see Table 1 in [31]).

When applying the theorem, *likelihoods* are computed from observed frequencies, then normalized to create probabilities (this normalization cancels out $P(E)$ in 3.5, so it need not be computed). For example, after $K = 10,000$ runs are divided into 1,000 lowest 10% *best* solutions and 9,000 *rest*, the range $rely = vh$ might appears 10 times in the *best* solutions, but only 5 times in the *rest*. Hence:

$$
\begin{aligned}
E &= (reply = vh) \\
P(best) &= 1000/10000 = 0.1 \\
P(rest) &= 9000/10000 = 0.9 \\
freq(E|best) &= 10/1000 = 0.01 \\
freq(E|rest) &= 5/9000 = 0.00056 \\
like(best|E) &= freq(E|best) \cdot P(best) = 0.001 \\
like(rest|E) &= freq(E|rest) \cdot P(rest) = 0.000504 \\
P(best|E) &= \frac{like(best|E)}{like(best|E) + like(rest|E)} = 0.66
\end{aligned} \tag{3.6}
$$

Previously [22] we have found that 3.6 is a poor ranking heuristic since it is distracted by low frequency evidence. For example, note how the probability of E belonging to the best class is moderately high even though its support is very low; i.e. $P(best|E) = 0.66$ but $freq(E|best) = 0.01$.

To avoid such unreliable low frequency evidence, we augment 3.6 with a support term. Support should *increase* as the frequency of a range *increases*, i.e. $like(x|best)$ is a valid support measure. STAR hence ranks ranges via

$$
P(best|E) * support(best|E) = \frac{like(x|best)^2}{like(x|best) + like(x|rest)} \tag{3.7}
$$

3.3 The STAR Algorithm

To apply 3.7, STAR runs in six phases. In terms of standard machine learning theory, step 1 generates a training set; steps 2,3,4 do some generalizations; and step 5 tests the learned theory on data not seen during training.

Incremental decision making is an important property of STAR. Observe how, in Figure 4.7, 50% of the score reduction arises from around 15% of the decisions. If a manager cannot implement all STAR's recommendations and a 50% reduction is adequate, she might elect to use just

these top 15% decisions. That is, STAR not only makes R recommendations, it also reports on the value of just applying just some subset of $r \subseteq R$.

1. *SAMPLE:* To sample the ranges from the models, STAR runs the simulated annealer K_1 times. Note that here, we sample across the ranges of all the attributes. While most of the time we sample randomly across the range, we also have a heuristic optimization called extreme sampling. This form of sampling works in the following manner: for x% (x is set to 5 by default), STAR samples only the extremums of the attributes. SEESAW, presented in subsubsection 2.3.1, was inspired by the effectiveness of this optimization, and implements it 100% of the time.

2. *DISCRETIZE:* The data seen in the K_1 samples is then discretized into $D = 10$ bins. Discretization converts a continuous range into a histogram with n break points $b_1 \ldots b_n$ where $\big(\forall i < j : b_i \leq b_j\big)$. After discretization, many observations can fall into the same range between b_i and b_{i+1} at frequency counts c_i. This study used equal width discretization; i.e.

$$\forall i, j : (b_i - b_{i-1}) = \big(b_j - b_{j-1}\big)$$

3. *CLASSIFY:* The ranges are then classified into those seen in BEST% *best* or *rest*.

4. *RANK:* The ranges are then ranked in increasing order using Support-Based Bayesian Ranking §3.2.

5. *PRUNE:* Also called the back select stage. STAR runs K_2 experiments with the models where the top ranked ranges $1..X$ ranges are pre-set and the remaining ranges can be selected at random.

6. *REPORT:* STAR returns the $1..X$ settings that optimize the best for the fitness function being used according to the weights applied to effort, defects, development time, and threats. These

settings are determined by iterating back from the minimum point achieved towards the first point that is statistically similar to the minimum point. This statistical difference is tested via a standard t-test.

To run our experiments, we had to apply our engineering judgment to set the parameters. The following are the default values:

$$K_1 = 10,000, K_2 = 1,000, D = 10, BEST = 10\%$$

STAR is very much a multi-purpose tool that not only has inspired others tools, but was also used in order to explore many problems in the software engineering world. Such problems include the following:

- the importance of automated software engineering (ASE) tools
- the ability to estimate effort without the presence of model calibration data
- the efficacy of drastic change versus improving certain aspects of a project with in the projects preset limits
- addressing the better-faster-cheaper, "pick any two", problem and its implications
- the future of software engineering and exploring what policies are going to be needed

Using STAR we will attempt to address these problems and discuss some of the solutions that STAR suggests.

3.4 Standard NASA Case Studies and Policies

STAR was tested on the four NASA case studies of Figure 3.1:

	ranges			values	
project	feature	low	high	feature	setting
OSP: Orbital space plane	prec	1	2	data	3
	flex	2	5	pvol	2
	resl	1	3	rely	5
	team	2	3	pcap	3
	pmat	1	4	plex	3
	stor	3	5	site	3
	ruse	2	4		
	docu	2	4		
	acap	2	3		
	pcon	2	3		
	apex	2	3		
	ltex	2	4		
	tool	2	3		
	sced	1	3		
	cplx	5	6		
	KSLOC	75	125		
JPL flight software	rely	3	5	tool	2
	data	2	3	sced	3
	cplx	3	6		
	time	3	4		
	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	7	418		

	ranges			values	
project	feature	low	high	feature	setting
OSP2	prec	3	5	flex	3
	pmat	4	5	resl	4
	docu	3	4	team	3
	ltex	2	5	time	3
	sced	2	4	stor	3
	KSLOC	75	125	data	4
				pvol	3
				ruse	4
				rely	5
				acap	4
				pcap	3
				pcon	3
				apex	4
				plex	4
				tool	5
				cplx	4
				site	6
JPL ground software	rely	1	4	tool	2
	data	2	3	sced	3
	cplx	1	4		
	time	3	4		
	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	11	392		

Figure 3.1: The four NASA case studies. Numeric values {*1, 2, 3, 4, 5, 6*} map to {*very low, low, nominal, high, very high, extra high*}.

- OSP: The Orbital Space Plane GNC prototype (a 1990s NASA system). OSP was an early prototype for OSP2.
- OSP2: the guidance and navigation control system of a current NASA launch vehicle, under development.
- FLIGHT/GROUND: these two are project templates used for nasa projects.

In Figure 3.1, *values* are *fixed* while *ranges* represent a space of options. The following observation can be made concerning the above projects: OSP2 constrains most of its features to fixed values while OSP allows more variation in feature ranges. Flight and Ground are more like project templates that, for the most part, enforce restrictions on the ranges of the project. OSP and OSP2 and both considered "Flight" projects.

		strategic?	tactical?
scale	prec: have we done this before?	✓	
factors	flex: development flexibility		✓
(exponentially	resl: any risk resolution activities?		✓
decrease	team: team cohesion		
effort)	pmat: process maturity	✓	
upper	acap: analyst capability	✓	
(linearly	pcap: programmer capability	✓	
decrease	pcon: programmer continuity	✓	
effort)	apex: analyst experience	✓	
	plex: programmer experience		✓
	ltex: language and tool experience		✓
	tool: tool use	✓	✓
	site: multiple site development	✓	✓
	sced: length of schedule		✓
lower	rely: required reliability		
(linearly	data: secondary memory storage requirements		
increase	cplx: program complexity	✓	✓
effort)	ruse: software reuse	✓	✓
	docu: documentation requirements	✓	✓
	time: runtime pressure		
	stor: main memory requirements		
	pvol: platform volatility		✓
hline COQUALMO	aa: automated analysis	✓	✓
defect removal	ett: execution-based testing tools	✓	✓
methods	peer: peer reviews	✓	✓

Figure 3.2: The policy classifications of the variables of COCOMO, COQUALMO, and the THREAT model.

In addition to the above, the policies need to be defined. The last two columns of Figure 3.2 show the results of Delphi panel session at JPL where the COCOMO variables were classified into those *tactical* variables that can be changed within the space of one project, and those *strategic* variables that required higher-level institutional change (and so may take longer to change). This effectively defines 2 policies: *strategic* and *tactical*. For example, the panel declared that *pmat* (process maturity) is hard to change within the space of a single JPL project. Specifying the use of one policy or the other restricts the attributes explored in the back select "prune" part of the STAR algorithm, and hence restricts the policies that can be used. These two analysis policies are used in the following sections for different experiments. When the policy is not specified, this inherently means that the experiment is using default analysis, i.e. all the attributes are being explored during the pruning phase.

Model	STAR Median	STAR Spread	Refrence Model	Differece	Difference%
COCOMO II	677.83	159.75	840.3	-162.47	-19.33
COQUALMO	540.39	313.22	417.25	+123.14	+29.51
Months	23.97	2.52	24.2	-0.23	-0.95
Threat	0	0	0	0	0

Figure 3.3: Sanity Checks: Showing the differences in estimates produced by STAR and the reference models

3.5 Sanity Checks

For the purpose of verifying that our system is valid in its estimations, we need to baseline it against the reference implementations of the models that we are using to make sure that we obtain ballpark estimates at least. For this purpose, a "Sanity" project was created that has all the effort multipliers set to nominal, the scale factors set to extremely high, and the defect removal tools left unset to allow STAR to be able to run. Note that these tools do not effect the COCOMO II effort (person-months), the Months (development time), or the Threat estimates. For comparisons with the reference COQUALMO model, the settings for the defect removal was set to the policy produced by STAR. Figure 3.3 presents the results of this exercise, where the last column presents the value of the percent difference between the Median result that STAR produces and the reference results. Note that the reference implementations are an online form [1] for COCOMO II, threat and Months and a USC Center of software engineering sourced excel file for COQUALMO.

Note that the results for STAR aren't set in stone simply because of the stochastic nature of the search that is conducted with in it, and because of the internal variation in model slopes that is conducted within the model implementations in STAR. Inspecting the results of the above experiment, we can see that the differences aren't large enough to warrant concern, keeping in mind that we aren't adhering to the calibrated models that the reference implementations are.

[1] http://sunset.usc.edu/research/COCOMOII/expert_cocomo/expert_cocomo2000.html

Policy	% Used	Policy	% Used
acap = 5	100	stor = 4	70
apex = 5	100	data = 3	60
flex = 6	100	time = 3.5	60
ltex = 4	100	data = 2.5	50
pcon = 5	100	cplx = 3.5	40
plex = 4	100	*peer* = 6	40
pmat = 3	100	stor = 3.5	40
rely = 5	100	pmat = 2.5	30
resl = 6	100	data = 2	20
site = 6	100	docu = 1.5	20
team = 6	100	pvol = 2.5	20
ett = 6	90	pvol = 4	20
pcap = 5	90	rely = 4.5	20
prec = 6	90	ruse = 2.5	20
aa = 6	80	pvol = 3	10
time = 4	80		

Figure 3.4: Stability of the policies produced for running the flight project.

3.6 Stability and Performance

In addition to the necessity of verifying that the results of STAR are within the ballpark of standardized methods, we also need to verify that we have some degree of stability. For this purpose we use two sample projects: one that is highly constrained (OSP) and one that is loosely constrained (flight). STAR was run ten times for both, and then the policy results produced were compared. Each of the projects were run through STAR ten times while using the *all* strategy, where all the model features are included in the search to produce policies. Figure 3.4 and Figure 3.5 below presents the percentage of times that a certain policy appears: the higher the percentage, the more that is indicative of the stability of that policy.

As we can see, there is a high rate of stability with respect to the policies being produced by STAR for these two sample projects. Any degree of instability that is indicated is a normal occurrence given that the core algorithm used in STAR is a meta-heuristic search algorithm, and also given that we are actively varying the internal parameters of the models used.

In addition to analyzing stability of the output of STAR in terms of the policies produced, we can also analyze the stability in terms of the model estimates produced by STAR for the projects,

Policy	% Used	Policy	% Used
aa = 6	100	pcon = 2.5	60
acap = 3	100	prec = 1.5	60
apex = 3	100	team = 2.5	60
ett = 6	100	tool = 2.5	60
flex = 5	100	apex = 2.5	40
ltex = 4	100	time = 3	40
pcon = 3	100	aa = 5.5	30
peer = 6	100	acap = 2.5	30
pmat = 4	100	stor = 3.5	30
prec = 2	100	docu = 4	20
resl = 3	100	ett = 5.5	20
team = 3	100	ruse = 2	20
tool = 3	100	sced = 2.5	20
cplx = 5.5	90	docu = 2.5	10
cplx = 5	80	ltex = 3.5	10
pmat = 3.5	80	resl = 2.5	10
sced = 2	80	sced = 3	10
ruse = 2.5	70	time = 4	10
time = 3.5	70		

Figure 3.5: Stability of the policies produced for running the OSP project.

as well as the time performance of STAR. Note that these are times produced on a machine running Ubuntu 8.04 with a 2.2GHz Intel Core 2 Duo processor and 2GB of RAM.

In Figure 3.6, "MinMed" indicates the median value of the corresponding estimate at the policy point, while "MinSP" indicates the spread of the corresponding estimate. Looking at Figure 3.6, we present the medians and the spreads of both the medians and the spreads of the estimates produced at the policy points in STAR. As we can see above, the spreads of the medians are with in acceptable limits, indicating a stability in estimates produced despite us varying the internal parameters of the models. In addition, the performance of STAR is quick thanks to fine tuning the parameters of the search. With regards to performance, we can also observe a correlation between how much a project is constrained and how long it takes to produce results. While the performance of the search part of the algorithm, indicated by "SATime", is almost the same for both projects, the total time is shorter for *OSP*. This suggests that the ranking and backselect times are dependent on how constrained a project is. Note that in either case, STAR is achieving in under a minute what took XOMO up to 30 minutes.

project	**Flight**										*Stats*	
run	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	**MEDIAN**	**SPREAD**
SATime(sec)	1.18	1.32	1.48	1.15	1.13	1.15	1.22	1.27	1.13	1.28	1.2	0.08
TotalTime(sec)	13.6	14.75	14.52	14.05	14.19	14.1	13.81	14.46	13.93	13.86	14.08	0.39
numPol	15	20	14	30	27	27	13	16	22	21	20.5	6.5
attNumber	15	19	14	23	22	22	13	16	19	18	18.5	3.5
MinMedEffort	101.79	86.69	115	51.12	78.88	61.74	111.7	97.54	81.9	99.88	92.11	9.68
MinSpEffort	56.82	49.52	71.55	29.24	41.9	36.09	61.22	65.83	51.32	64.39	54.07	10.32
MinMedDefects	147.23	24.46	168.6	28.52	34.07	22.6	101.8	66.61	42.9	66	54.45	47.35
MinSpDefects	117.25	16.85	129.41	16.88	23.11	14.97	79.06	42.5	31.32	44.82	36.91	42.15
MinMedThreat	0	0	0	0	0	0	0	0	0	0	0	0
MinSpThreat	0	0	0	0	0	0	0	0	0	0	0	0
MinMedMonths	13.97	13.44	15.54	11.55	12.96	12.08	14.4	14.22	13.09	14.48	13.7	0.69
MinSpMonths	2.39	2.23	2.94	1.73	2.13	2.08	2.41	2.79	2.45	2.72	2.4	0.32

project	**OSP**										*Stats*	
run	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	**MEDIAN**	**SPREAD**
SATime(sec)	1.21	1.47	1.21	1.25	1.17	1.2	1.14	1.21	1.22	1.2	1.21	0.01
TotalTime(sec)	8.98	8.86	8.78	8.88	8.66	8.39	8.84	8.74	8.62	8.79	8.79	0.07
numPol	29	20	20	16	27	17	29	21	25	27	23	4
attNumber	19	17	16	16	18	16	18	17	18	18	17.5	0.5
MinMedEffort	516.9	578.42	613.46	564.39	658.75	499.31	648.82	575.46	722.7	543.94	576.94	71.88
MinSpEffort	116	140.28	137.78	159.23	159.14	109.52	131.51	138.18	153.13	126.3	137.98	15.15
MinMedDefects	307.68	249.06	263.7	259.33	215.12	205.38	335.27	245.82	247.84	259.82	254.2	9.51
MinSpDefects	80.7	69.39	79.47	74.39	52.58	50.47	91.32	75.22	66.13	72.06	73.22	6.25
MinMedThreat	3.36	3.42	6.96	5.25	2.6	0	3.46	2.87	4.17	0	3.39	0.78
MinSpThreat	0.33	6.22	4.61	3.28	1.15	0	6.26	3.15	6.63	3.45	3.36	2.86
MinMedMonths	22.64	24.05	24.29	29.28	27.19	23.38	25.09	28.33	25.71	23.23	24.69	2.5
MinSpMonths	2.44	2.83	2.97	3.53	3.4	2.86	2.73	3.03	2.86	2.37	2.86	0.17

Figure 3.6: Performance statictics of STAR runs on the Flight and OSP projects.

3.7 STAR vs. LC

LC is a standard tuning/estimation method presented in §2.2. This method is dependent on local data to generate the tunings and estimates. We seek to investigate whether STAR can generate estimates of similar quality to LC. If this is achieved, it will allow us to bypass the the data drought issue, while at the same time adding validation to using AI search for software process control. Note that the same study was done by Menzies et. al [59].

3.7.1 Experiments

Figure 3.1 shows various *Projects* expressed in term of $float$ing and $fixed$ variables. For example, with JPL's flight systems, the *rely* (required reliability) can float anywhere in the upper range; i.e. $rely \in \{3,4,5\}$. However, for flight systems, *sced* (schedule pressure) is tightly defined (so *sced* is

fixed to the value 3).

For this section, we will be looking at the four case studies of Figure 3.1, plus a fifth study called "ALL" that uses the entire range COCOMO attributes, unconstrained by a particular project specification. Each study was repeated twice- one for controlling just the strategic variables and once for controlling just the tactical variables. This resulted in ten experiments. The results of the experiments are presented later in Chapter 4. for the purposes of this section, all that was needed were simulations that were run at the policy point of the particular case. The results of these simulations were used to create the STAR MRE graphs.

For each of the cases, the following procedure was repeated 20 times. Ten examples were removed at random and Boehm's local calibration (LC) procedure [11, p526-529] was used to train a COCOMO model on the remaining *Project* examples. LC's estimates were then compared to the estimates generated by STAR's simulation at the policy point (i.e. floating over both the policy and the *Model* ranges). Figure 3.8 show the median difference in the estimates generated by LC and STAR . Note that, in all the cases, the difference is under 35%. This result is confirmed by the plots in Figure 3.7, where it is evident that for all the cases, the MRE plots for star are close to the corresponding plots for LC. Also notice that, for all the plots, STAR's MRE never exceeds 100%. This shows that the difference between STAR and LC is never excessively high, where STAR generates estimates in the ball park of LC's estimates.

3.7.2 Discussion

The Above result presented by STAR is very interesting in that, even without any calibration data, and with only a realization of the space of those tunings, STAR was able to generate estimates similar to the estimates generated by LC. Keep in mind the LC works with local tuning data, and really is expected to do better. This result introduces us to the ability and effectiveness of STAR, and search based methods, to manage uncertainty in the absence of local data.

How are we to explain the remarkable effectiveness of STAR in managing uncertainty? Re-

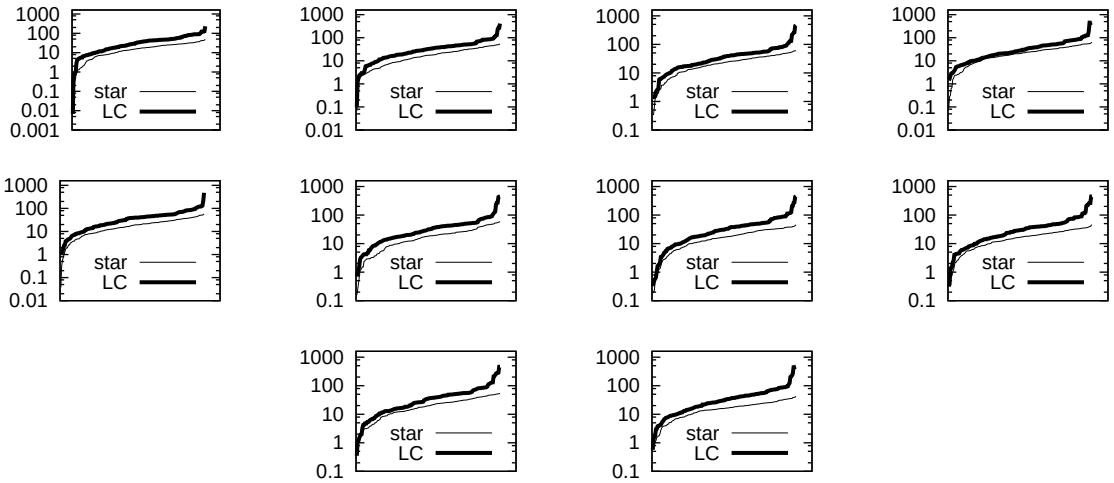


Figure 3.7: Plots of the MREs of the effort estimates generated by LC (fat line) and STAR (thin line). The ten plots are sorted in order of the median differences (shown in Figure 3.8). Within in each plot, the results are sorted by the MRE of the estimated offered by LC or STAR. A log scale is used for the y-axis.

case study	control method	δ %
OSP	strategic	21.78
ground	tactical	22.17
All	strategic	22.95
All	tactical	23.14
ground	strategic	23.38
flight	tactical	25
OSP2	strategic	26.6
OSP	tactical	27.71
flight	strategic	28.47
OSP2	tactical	32.07

Figure 3.8: Median $\delta = (estimate(STAR) - estimate(lc))$ between effort estimates generated by conventional means (LC) and STAR.

searchers in planning and theorem proving have recently shown that as model complexity grows, other constraining effects may appear such as "master variables"; i.e. a small number of settings that control all other settings [26, 79]. Such master variables can greatly reduce the search space within large models.

We hypothesize that software process models also contain master variables; i.e. much much of uncertainty in a model is due to the influence of a small subset of model variables. If so, then after (a) ranking variables by their ability to constrain the output; and (b) applying a small number of the

top-ranked variables; then it should be possible to (c) make stable predictions in the constrained space.

Chapter 4

NASA Experiments and Advanced ASE Tools

In this chapter, we will present the anti-automation bias present in the software engineering community. This is followed by a study that attempts to argue against this bias by presenting a business case for using automated software engineering (ASE) tools. Note that this study was initially conducted for and reported in **ASE'07**: *The business case for automated software engineering*. [60]

4.1 Sociology vs. Tools

Much current ASE research concerns *automatic analysis* of source code or better *execution-based testing tools*, These tools might, say, verify formal properties or search for the fewest regression tests that exercise most of the system. Some of these tools are ready for industrial use; e.g, SPIN [44] or JPF [42], just to name a few.

We seeked to make the business case for introducing some of these new ASE tools into large NASA projects. This case proved difficult to make, due to an *anti-automation bias* and the *local tuning problem*.

The *anti-automation bias* was seen at an ICSE 2007 panel. Tim Lister (a co-author of Peopleware [29]) commented that "sociology beats technology in terms of successfully completing projects"- a notion endorsed by the other panelists. That is, software project managers should focus *less* on new ASE tools and *more* on managing the sociology aspects of their team (e.g. decrease staff turnover).

Figure 4.1 offers some support for this bias. This figure shows the known relative productivity effects of changing project features. According to this figure, the benefits of automatic tools (ranked number nine in the list) can be equaled or bettered via other means (e.g. any item 1 to 8 *or* any *pair* of items 10 to 22).

Note that this support for the anti-automation bias is based solely on the development effort; i.e. Figure 4.1 is blind to the impact of new ASE tools in reducing *defects* and any other *threats* to the success of the project. A complete business case should therefore study predictors for effort *and* defects *and* threats *and* schedule.

id	features	relative weight
1	Personnel/team capability	3.53
2	Product complexity	2.38
3	Time constraint	1.63
4	Required software reliability	1.54
5	Multi-site development	1.53
6	Doc. match to life cycle	1.52
7	Personnel continuity	1.51
8	Applications experience	1.51
9	Use of software tools	1.50
10	Platform volatility	1.49
11	Storage constraint	1.46
12	Process maturity	1.43
13	Language & tools experience	1.43
14	Required dev. schedule	1.43
15	Data base size	1.42
16	Platform experience	1.40
17	Arch. & risk resolution	1.39
18	Precedentedness	1.33
19	Developed for reuse	1.31
20	Team cohesion	1.29
21	Development mode	1.32
22	Development flexibility	1.26

Figure 4.1: Relative effects of COCOMO attributes on development effort. Data from a regression analysis of 161 projects [12].

4.2 NASA Experiments and ASE Tools

Before we can assess the relative merits of new ASE tools versus other methods, we must first define "new ASE tools" and "other methods". Such a definition can be achieved via the ontology of COCOMO, COQUALMO, SCHEDULE, and THREAT, shown in Figure 2.1, and discussed in §2.1. This figure lists a variety of project *features* with the *range* {very low, low, nominal, high, very high, extremely high} or

$$\{vl = 1, l = 2, n = 3, h = 4, vh = 5, xh = 6\}$$

Lister's sociological features occur many times within Figure 2.1. For example, *team* refers to the sociology within a development team and its cohesiveness while *pcon* refers to the staff turnover rate within a project. Also listed in Figure 2.1 are factors like *acap, pcap* representing analyst and programmer capabilities (respectively).

As to technological features, new ASE tools appear as *execution-based testing tools (ett)* and *automated analysis (aa)*. Chulani [30] defines the top half of *automated analysis* as:

4 (**high**): intermediate-level module and inter-module code syntax and semantic analysis. Simple requirements/design view consistency checking.

5 (**very high**): More elaborate requirements/design view consistency checking. Basic distributed-processing and temporal analysis, model checking, symbolic execution.

6 (**extremely high**): Formalized[1] specification and verification. Advanced distributed processing and temporal analysis, model checking, symbolic execution.

The upper half of *execution-based testing and tools* is:

4 (**high**): Well-defined test sequence tailored to organization (acceptance / alpha / beta / flight / etc.) test. Basic test coverage tools, test support system.

[1] Consistency-checkable pre- conditions and post-conditions, but not necessarily mathematical theorems.

5 (**very high**): More advanced test tools, test data preparation, basic test oracle support, distributed monitoring and analysis, assertion checking. Metrics-based test process management.

6 (**extremely high**): Highly advanced tools for test oracles, distributed monitoring and analysis, assertion checking Integration of automated analysis and test tools. Model-based test process management.

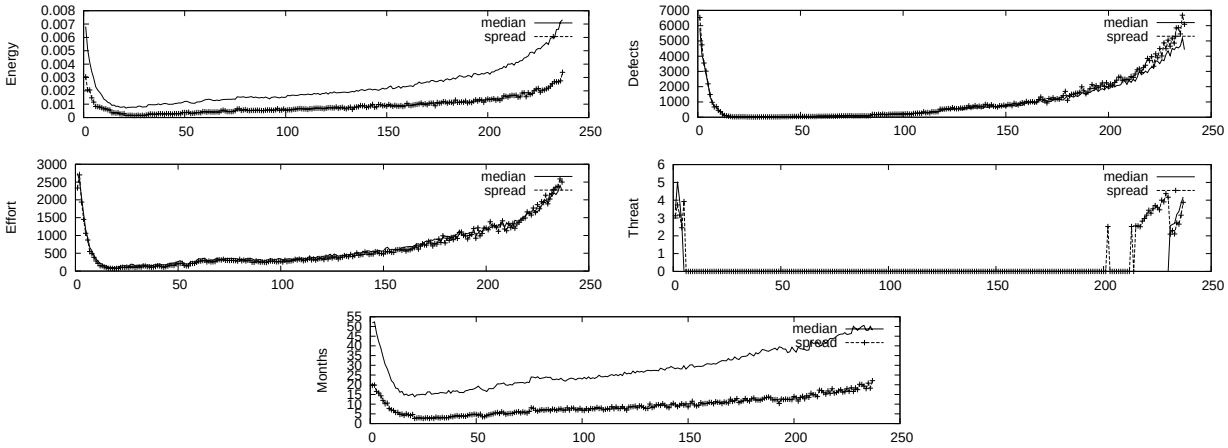
A review of recent proceedings of the IEEE ASE conferences suggests that a range of five or six in the above features includes the kind of "new ASE tools" we mean to explore in this section. Hence, to compare "new ASE tools" to "other methods", we will try to reduce effort and defects and development time and threats using just

$$\textit{automated analysis} \in \{5,6\} \vee$$
$$\textit{execution-based testing and tools} \in \{5,6\}$$

or "other methods" (i.e. other ranges of Figure 2.1).

We ran STAR over the projects specified in Figure 3.1 using the default, strategic and tactical (defined in Figure 3.2 and §3.4) policies in defining controlable model features. Note that the *ALL* project is basically defined as an open ended project with no preset limits on the ranges or values of the model attributes. Hence, the ranges for all the default ranges of the models, defined in Figure 2.13.

The lists at the bottom of Figure 4.2 through to Figure 4.16 show how Equation 3.7 ranked the various project decisions that, according to STAR and the fitness function expressed in Equation 3.2, achieve the best balance of reducing the various model estimates given the used weights. These ranks correspond to the X-axis of the plots at the top of that figure. "Median" plots the

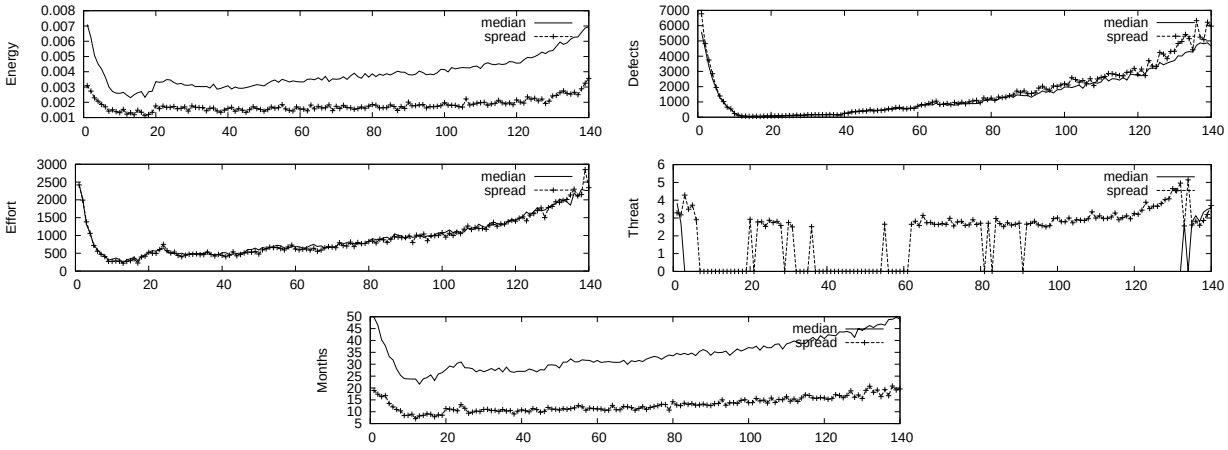


X	feature = range
1	rely = 5
2	pcap = 5
3	acap = 5
4	ltex = 5
5	pmat = 6
6	site = 6
7	resl = 6
8	flex = 6
9	team = 6
10	plex = 5
11	prec = 6
12	automated-analysis = 6 ▲▲
13	apex = 5
14	tool = 5
15	pcon = 5
16	execution-testing-and-tools = 6 ▲▲
17	time = 3.5
18	apex = 4.5
19	acap = 4
20	sced = 2.5

Figure 4.2: Default analysis: ALL results.

50% percentile of the defect/ effort/ schedule/ threat values seen after imposing ranges 1..X (and selecting all other ranges at random). "Spread" shows the 75%-50% percentile range. These plots are U-shaped:

- On the left-hand-side of each plot, poor results were seen after applying too few constraints. That is, models perform poorly if we do not control them enough.
- On the right-hand-side of each plot, poor results were also seen. In terms of Equation 3.7 , these are the ranges with low support and low probability of belonging to *best* class. Hence, it is not surprising that applying these superfluous constraints was counter-productive.



X	feature = range
1	acap = 5
2	pcap = 5
3	apex = 5
4	site = 6
5	tool = 5
6	pmat = 6
7	prec = 6

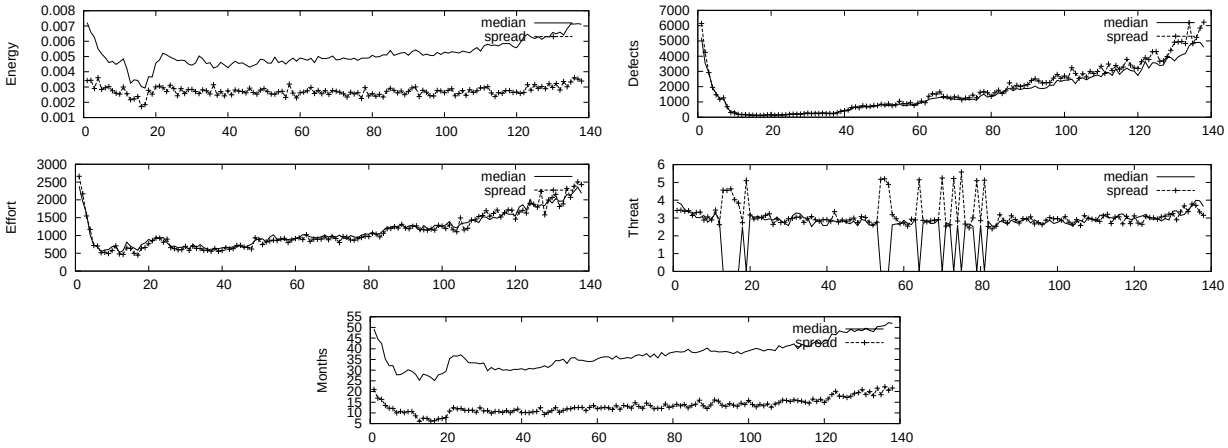
Figure 4.3: Strategic analysis: ALL results.

The policies that recommend new ASE tools in the plots are designated by the "▲ ▲" symbol next to them.

Observing the plots and the policies in Figure 4.2 through to Figure 4.16, there are several observations we can make. These are global observations which seem to apply across the board for every one

- For all the models, except threat, the plots have a U-shape with different degrees if concavity. For these plots, we can also notice that not only are the medians of the estimates being reduced, but that also the spreads are being reduced as well. Also, these spread curve never seem to excessively exceed the medians in any of the project/policy combinations. This implies that through STAR we are able to control the model estimates and consistently reduce them with varying degrees.

- The policies achieved automatically in STAR always include a very low percentage of all the possible recommendations in the search space. One example is Figure 4.4, where the policy



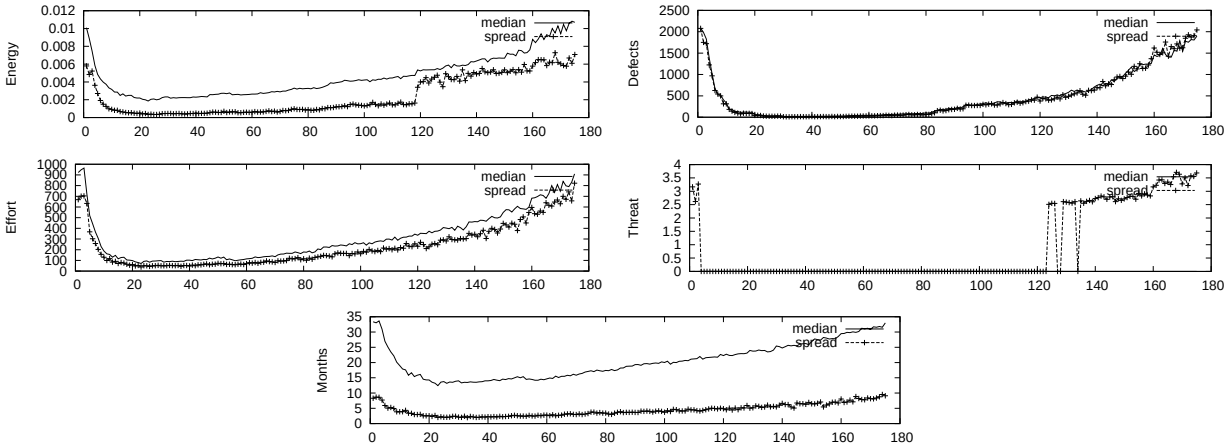
X	feature = range
1	resl = 6
2	ltex = 5
3	site = 6
4	plex = 5
5	tool = 5
6	flex = 6
7	automated-analysis = 6 ▲▲
8	peer-reviews = 6
9	pvol = 5
10	execution-testing-and-tools = 6 ▲▲
11	cplx = 1.5
12	sced = 2.5

Figure 4.4: Tactical analysis: ALL results.

contains 12 total recommendations. The search space in this case is about 140 recommendations, making the policy include about 8.6% of the total space of recommendations. This suggests that STAR is indeed capable of vastly reducing the space of possible recommendations that a manager can chose from.

- The initial rapid decrease in the estimate curves, compared to the gradual increase in these curves as superfluous recommendations are added to the list. This suggests that we are able to control, and initially drastically reduce (especially for less constrained projects), the model estimates by controlling a subset of the features in those models. This supports the concept of having "collar" features, which in turn supports our approach in selecting a subset of these features to be included in policy recommendations.

- The top ranked recommendations produced by STAR always have values that exist on ex-

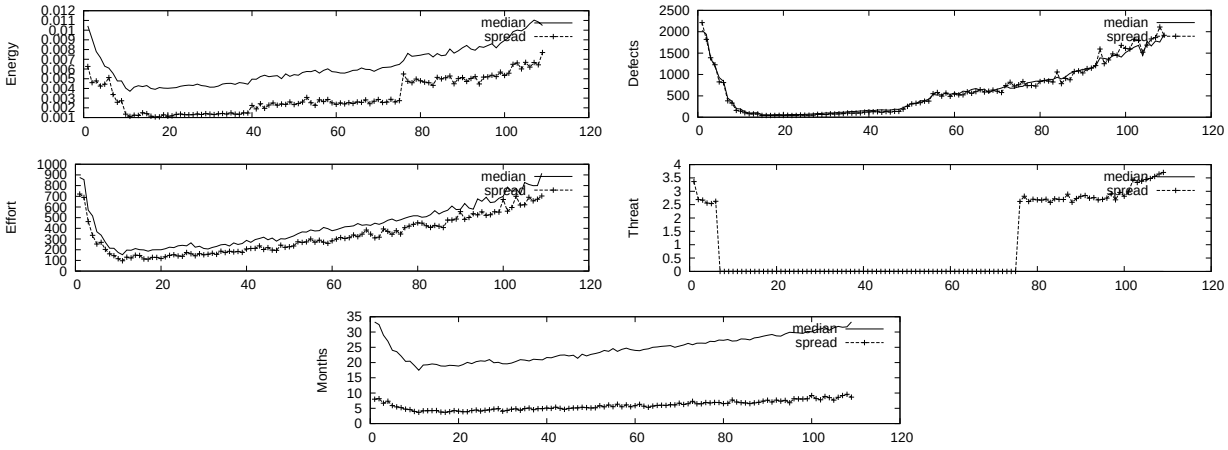


X	feature = range
1	pmat = 3
2	rely = 5
3	ltex = 4
4	site = 6
5	team = 6
6	plex = 4
7	pcap = 5
8	pcon = 5
9	acap = 5
10	automated-analysis = 6 ▲▲
11	stor = 3.5
12	resl = 6
13	data = 3
14	apex = 5
15	time = 4
16	stor = 4
17	flex = 6
18	time = 3.5
19	execution-testing-and-tools = 6 ▲▲
20	ruse = 2.5
21	data = 2.5
22	prec = 6

Figure 4.5: Default analysis: flight results.

tremum of the range specified by the project. Note that it is this phenomenon that inspired implementing extreme sampling (§3.3), as well as SEESAW [64].

Aside form the above general observations, we can also comment on the importance of advanced ASE tools. Note that, in all the policies except for one (that is for $\frac{14}{15}$ of the cases), advanced ASE tools are being recommended in one form or the other. It should also be noted that,

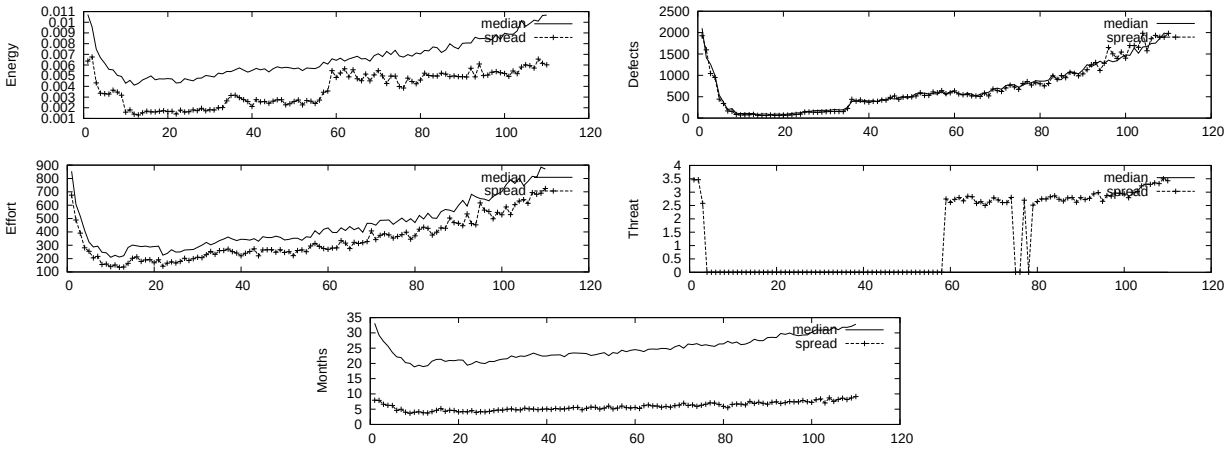


X	feature = range
1	pmat = 3
2	site = 6
3	pcap = 5
4	pcon = 5
5	acap = 5
6	automated-analysis = 6 ▲▲
7	apex = 5
8	execution-testing-and-tools = 6 ▲▲
9	ruse = 2.5
10	prec = 6

Figure 4.6: Strategic analysis: flight results.

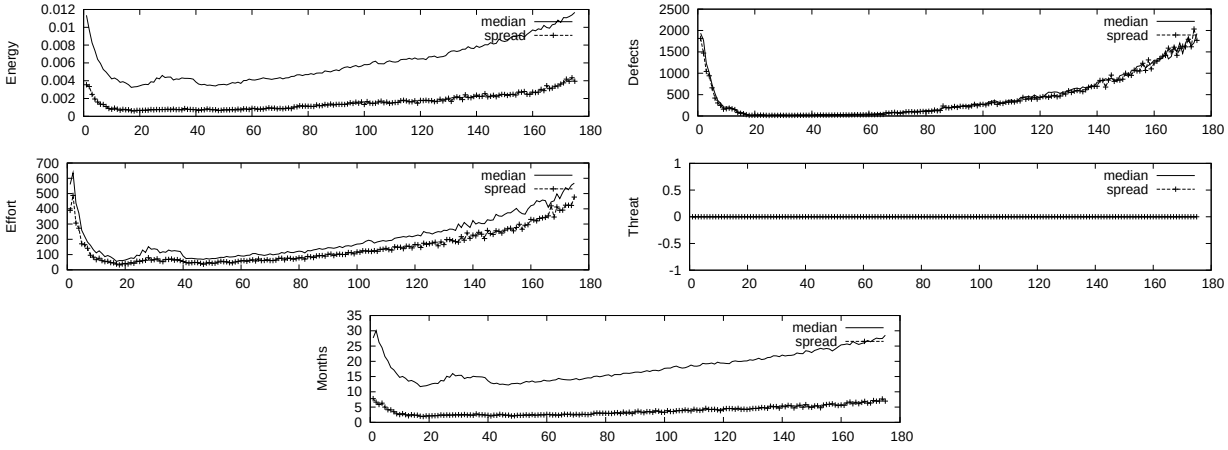
when ASE tools are recommended, they are always recommended at the highest levels available. This results asserts that, not only are these tools important, but that it is also necessary to apply them at their highest levels whenever they are recommended.

Another observation that can be made is that advanced ASE tools become more important as a project becomes more constrained. "Constrained" here can be interpreted as more mature and further down the development path, where more of the properties of the projects are buttoned down and set. This can be seen through the policies in the different case studies. One example is how advance ASE tools are ranked 6^{th} and 8^{th} in Figure 4.6 that presents that results of flight strategic, while they are ranked at 1^{st}, 2^{nd} and 3^{rd} in all the OSP2 case studies presented in Figure 4.14 to Figure 4.16.



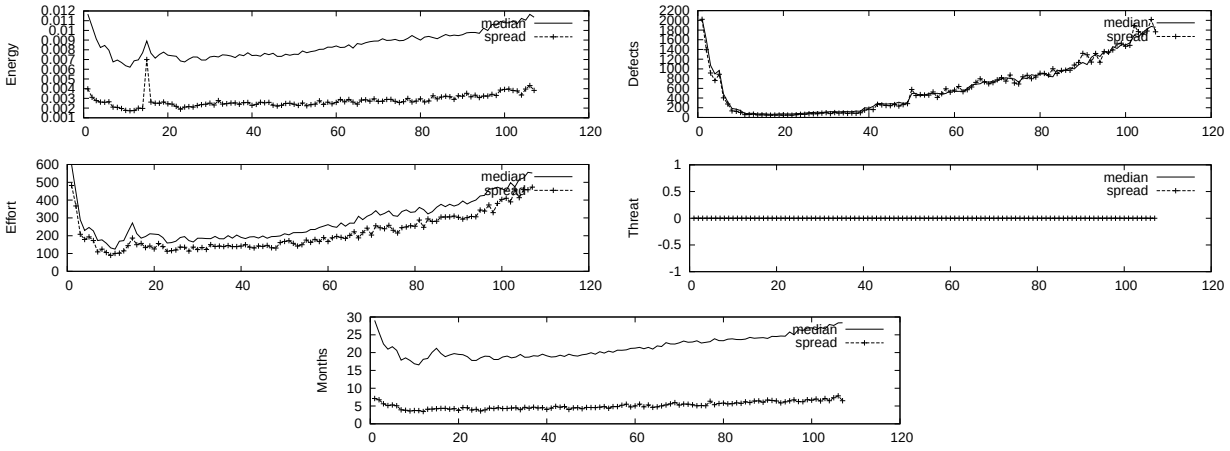
X	feature = range
1	site = 6
2	ltex = 4
3	plex = 4
4	execution-testing-and-tools = 6 ▲▲
5	resl = 6
6	automated-analysis = 6 ▲▲
7	flex = 6
8	peer-reviews = 6
9	ruse = 2.5
10	pvol = 4
11	cplx = 3.5

Figure 4.7: Tactical analysis: flight results.



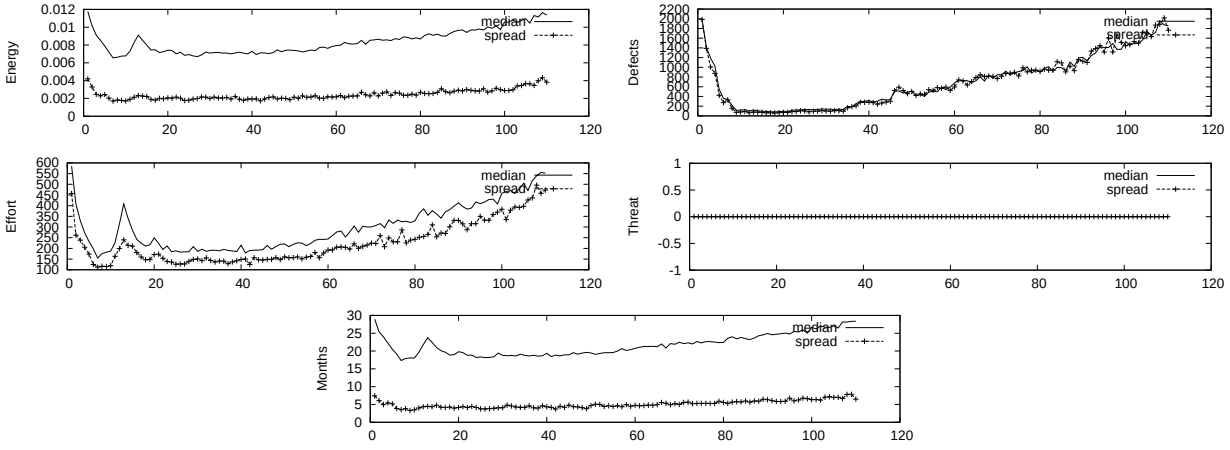
X	feature = range	
1	rely = 4	
2	site = 6	
3	ltex = 4	
4	pcon = 5	
5	team = 6	
6	resl = 6	
7	plex = 4	
8	prec = 6	
9	flex = 6	
10	stor = 4	
11	acap = 5	
12	pcap = 5	
13	peer-reviews = 6	▲▲
14	pmat = 3	
15	apex = 5	
16	automated-analysis = 6	▲▲

Figure 4.8: Default analysis: ground results.



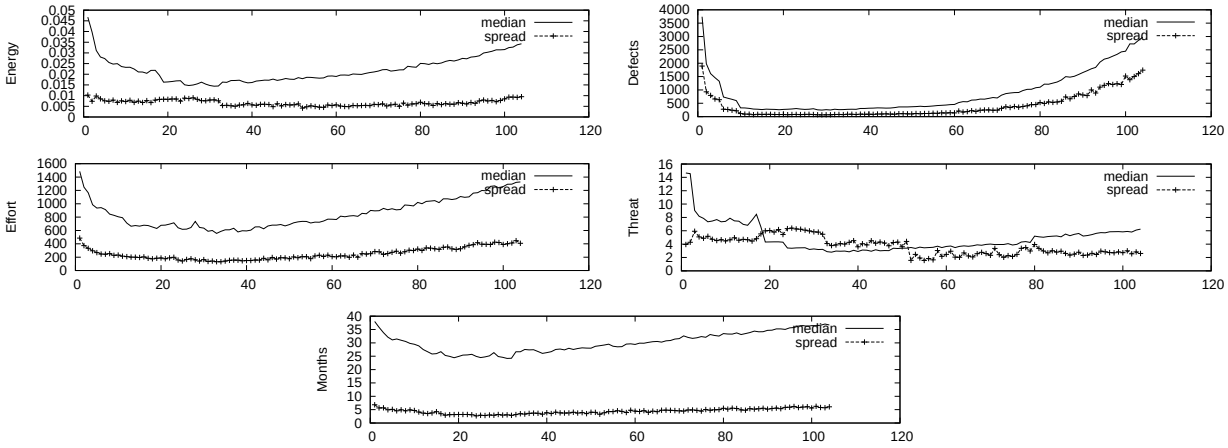
X	feature = range
1	pcon = 5
2	site = 6
3	apex = 5
4	pmat = 3
5	execution-testing-and-tools = 6 ▲▲
6	prec = 6

Figure 4.9: Strategic analysis: ground results.



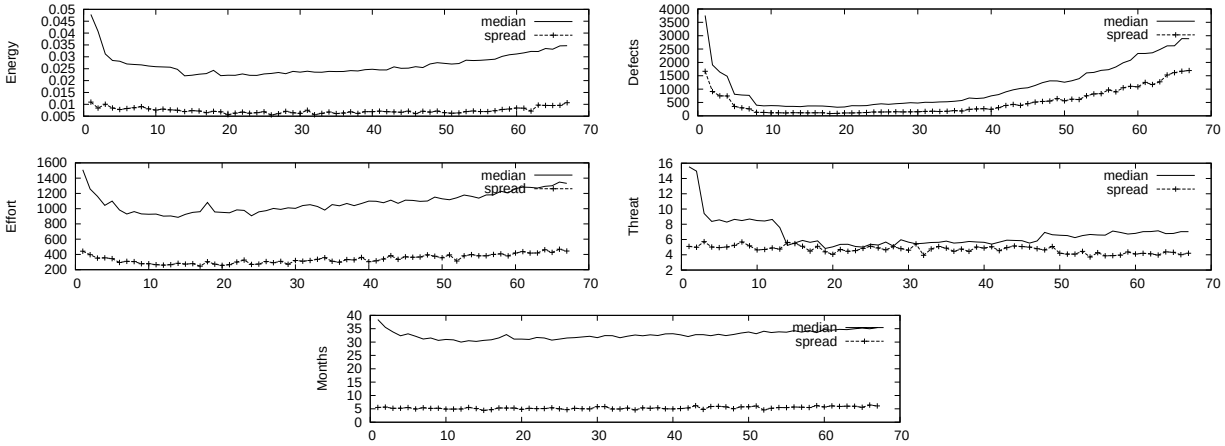
X	feature = range
1	site = 6
2	ltex = 4
3	plex = 4
4	execution-testing-and-tools = 6 ▲▲
5	resl = 6
6	flex = 6

Figure 4.10: Tactical analysis: ground results.



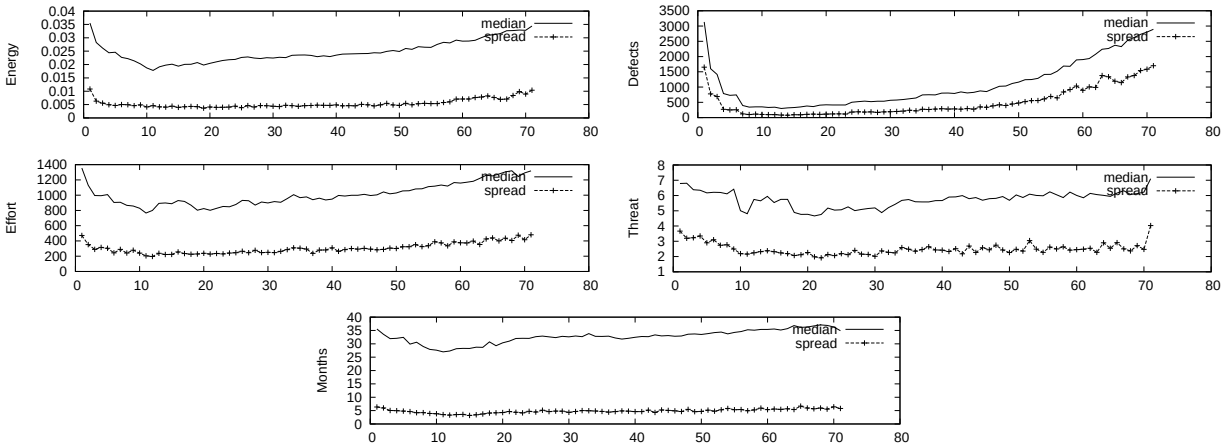
X	feature = range
	automated-analysis = 6 ▲▲
1	pmat = 4
2	ltex = 4
3	acap = 3
4	execution-testing-and-tools = 6 ▲▲
5	apex = 3
6	team = 3
7	pcon = 3
8	peer-reviews = 6
9	prec = 2
10	flex = 5
11	resl = 3
12	tool = 2.5
13	tool = 3
14	cplx = 5.5
15	sced = 2
16	cplx = 5
17	pmat = 3.5
18	acap = 2.5
19	team = 2.5
20	pcon = 2.5
21	prec = 1.5
22	time = 3.5

Figure 4.11: Default analysis: OSP results.



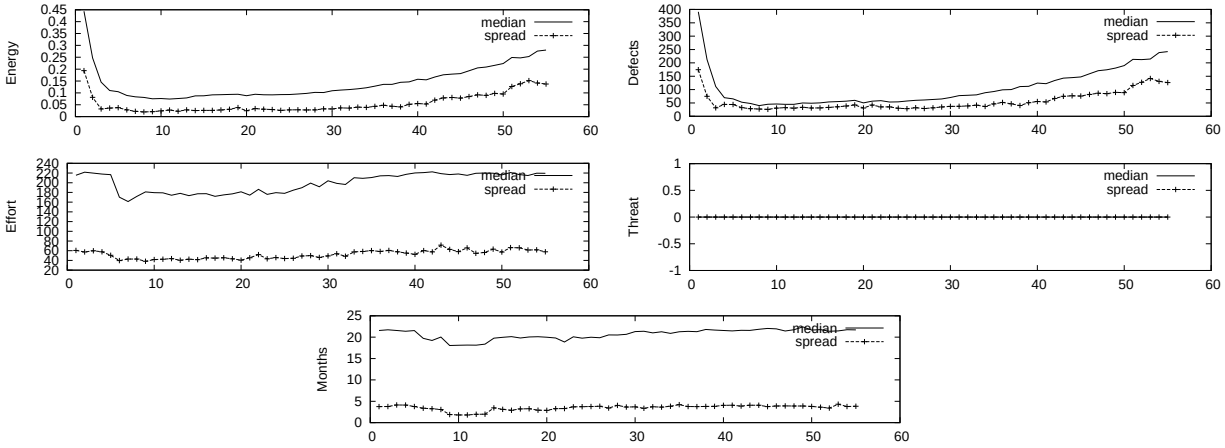
X	feature = range
1	automated-analysis = 6 ▲▲
2	pmat = 4
3	acap = 3
4	execution-testing-and-tools = 6 ▲▲
5	apex = 3
6	pcon = 3
7	peer-reviews = 6
8	prec = 2
9	tool = 2.5
10	tool = 3
11	cplx = 5.5
12	cplx = 5
13	pmat = 3.5

Figure 4.12: Strategic analysis: OSP results.



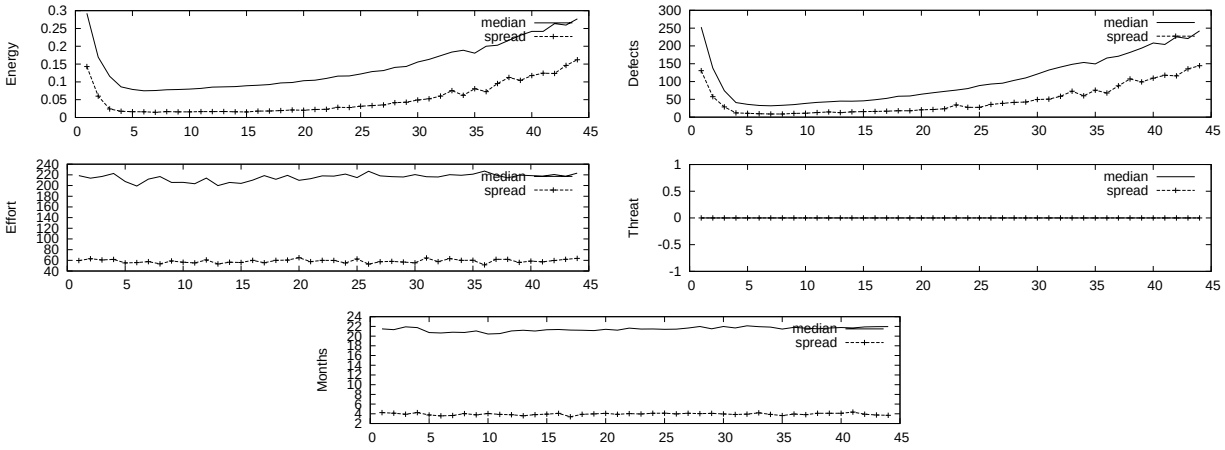
X	feature = range
	execution-testing-and-tools = 6 ▲▲
	ltex = 4
	automated-analysis = 6 ▲▲
	tool = 3
	flex = 5
	peer-reviews = 6
	resl = 3
	sced = 2
	cplx = 5
	ruse = 2.5

Figure 4.13: Tactical analysis: OSP results.



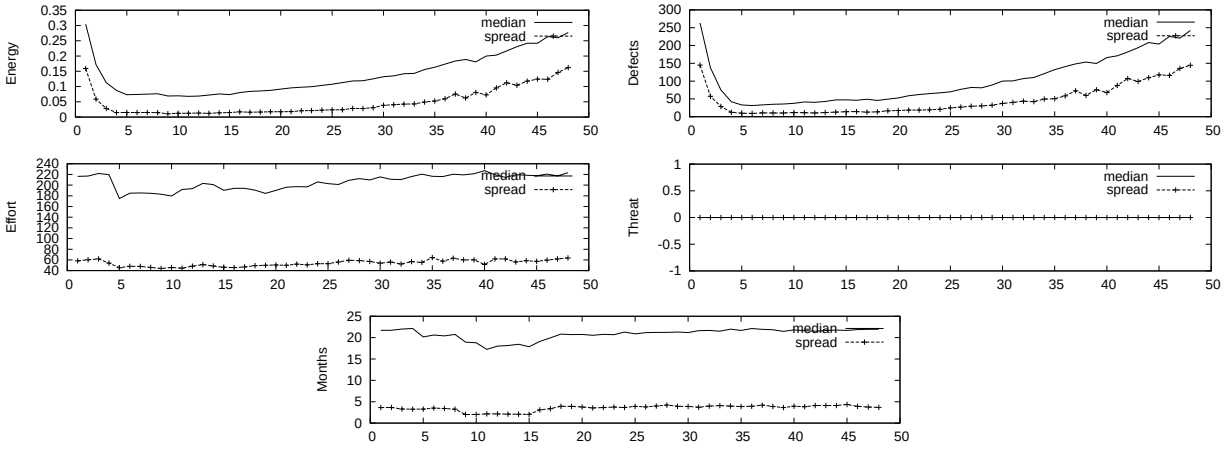
X	feature = range
1	automated-analysis = 6 ▲▲
2	execution-testing-and-tools = 6 ▲▲
3	peer-reviews = 6
4	pmat = 5
5	ltex = 5
6	prec = 5
7	docu = 4
8	sced = 2.5

Figure 4.14: Default analysis: OSP2 results.



X	feature = range
1	automated-analysis = 6 ▲▲
2	execution-testing-and-tools = 6 ▲▲
3	peer-reviews = 6
4	prec = 5
5	pmat = 5

Figure 4.15: Strategic analysis: OSP2 results.



X	feature = range
1	automated-analysis = 6 ▲▲
2	execution-testing-and-tools = 6 ▲▲
3	peer-reviews = 6
4	ltex = 5
5	docu = 4
6	peer-reviews = 5.5
7	docu = 3.5
8	sced = 3

Figure 4.16: Tactical analysis: OSP2 results.

project	ALL	OSP	OSP2	flight	ground
policies	a s t	a s t	a s t	a s t	a s t
acap = 3		• •			
acap = 4	•				
acap = 5	• •			• •	•
apex = 3		• •			
apex = 5	• •			• •	• •
aa = 6	• •	• • •	• • •	• • •	•
cplx = 2	•				
cplx = 4				•	
cplx = 5		• • •			
cplx = 6		• •			
data = 3				•	
docu = 4			• •		
ett = 6	• •	• • •	• • •	• • •	• •
flex = 5		• •			
flex = 6	• •			• •	• •
ltex = 4		• •		• •	• •
ltex = 5	• •		• •		
pcap = 5	• •			• •	•
pcon = 3		• •			
pcon = 5	•			• •	• •
peer = 6	•	• • •	• • •	•	•
plex = 4				• •	• •
plex = 5	• •				
pmat = 3				• •	• •
pmat = 4		• •			
pmat = 5			• •		
pmat = 6	• •				

Figure 4.17: Policies produced for the NASA projects, part 1

In addition to the above figures, we present Figure 4.17 and Figure 4.18. These two figures are what we called braille charts, and aim to summarize the results of the studies in a more concise fashion compared to the above figures. Using these charts, we can make the observation that, for effort and months (development time), the reductions produced for less constrained projects are much larger than those achieved for more constrained ones. In other words, we are able to exploit the uncertainty within these less constrained projects to drive large improvements. This supports the result expressed in [58], where it is stated that "if you fix everything, you have nothing else to fix". Defects reduction, on the other hand, are dramatic for all the cases, with the least reduction brings down defects to 26% of its initial value, which can be attributed to the use of advanced ASE tools.

Note that these studies seem to show that, even though the threat model seems to make sense

project	ALL			OSP			OSP2			flight			ground		
policies	a	s	t	a	s	t	a	s	t	a	s	t	a	s	t
prec = 2				•	•										
prec = 5							•	•							
prec = 6	•	•								•	•		•	•	
pvol = 4												•			
pvol = 5			•												
rely = 4													•		
rely = 5	•									•					
resl = 3				•		•									
resl = 6	•		•							•		•	•		•
ruse = 3						•				•	•	•			
sced = 2				•		•									
sced = 3	•		•				•		•						
site = 6	•	•	•							•	•	•	•	•	•
stor = 4										•			•		
team = 3				•											
team = 6	•									•			•		
time = 4	•			•						•					
tool = 3				•	•	•									
tool = 5	•	•	•												
models	percentage of original value														
effort	5	21	24	54	68	63	78	93	83	9	19	26	13	41	36
defects	1	21	4	10	12	12	17	15	14	3	10	6	4	26	19
threat	0	0	80	69	*108*	70	0	0	0	0	0	0	0	0	0
months	32	57	54	70	86	79	92	94	94	40	57	59	44	73	67

Figure 4.18: Policies produced for the NASA projects, part 2. This figure also presents the differences produced by the policies

theoretically in managing and estimating risk, empirically is seems too insensitive to model inputs, causing low (and usually non-existent) medians and high (and jumpy) spreads (threat never really exceeds 20). The reason behind this is possibly the look-up table nature of this model, where instead of progressive transitions, we have value jumps which are exhibited in a small area of the look-up tables as shown in Figure 2.12. Due to this behavior, the threat model will not be emphasized throughout this work. Future investigation into redefining or making this model more effective is possible, but is outside the scope of this work.

In conclusion, social factors are indeed very important. However, advanced ASE tools shouldn't be neglected, especially for more defined and constrained projects.

Chapter 5

Expert Studies

5.1 JPL Studies

The purpose of this study is to observe how policy recommendations of STAR would change for ground and flight projects at JPL over time. Through correspondence with Jarius Hihn from JPL we were able to obtain past and current project specifications of flight and ground software projects according to the COCOMO-II model. These specifications are indicated in Figure 5.1 through to Figure 5.4. Note that here the data ranges that are presented for the different JPL projects are based on actual project ranges from products that are merged together. However this is dependent on the on the amount of sample projects available, and instances like ground 2000 in Figure 5.3 are due to only having a single sample project for that type and time period. Aside from this exception, the general trend for these project templates seems to be that older projects are more restricted than newer ones, having more set values for attributes, as well as tighter ranges for some of these attributes. This is a case of "20-20 hindsight", where we know more about the more mature past projects than the more recent projects.

In addition to the above project model descriptions, we were also given modified policy spec-

project	ranges			values	
	feature	low	high	feature	setting
JPL flight software circa 2000	prec	2	3	docu	3
	flex	1	3		
	resl	3	4		
	team	3	4		
	pmat	3	4		
	rely	4	5		
	cplx	4	5		
	ruse	2	4		
	time	3	5		
	stor	3	4		
	acap	3	4.5		
	pcap	3	4		
	pcon	3	5		
	apex	3	4		
	plex	1	4.5		
	ltex	3	4		
	tool	2	4.5		
	site	2	6		
	Ksloc	10.21	294.39		

project	ranges			values	
	feature	low	high	feature	setting
JPL flight software circa 1990	prec	1	5	docu	3
	resl	2	3		
	team	2	6		
	pmat	2	4		
	rely	3	4.5		
	cplx	3	6		
	ruse	2	3		
	time	3	6		
	pvol	2	4		
	acap	2	5		
	pcap	3	5		
	pcon	1.5	5		
	plex	1	4		
	ltex	2	5		
	tool	1	3		
	site	2	6		
	docu	3	4		
	Ksloc	8	253.13		

Figure 5.1: JPL flight circa 1990 and 2000

project	ranges			values	
	feature	low	high	feature	setting
JPL flight software circa 1980	pmat	1	3	prec	2
	rely	4	5	flex	2
	cplx	4	5	resl	2
	data	2	5	team	3
	time	5	6	ruse	3
	stor	5	6	pcon	3
	pvol	2	4	site	4
	acap	3	5	docu	4
	pcap	3	4		
	apex	2	4		
	plex	1	4		
	ltex	2	4		
	tool	1	2		
	sced	3	5		
	Ksloc	1.5	233		

project	ranges			values	
	feature	low	high	feature	setting
JPL flight software circa 1970	rely	3	5	prec	2
	cplx	2	6	flex	2
	data	2	4	resl	2
	time	3	6	team	3
	stor	4	6	pmat	3
	pvol	2	4	ruse	3
	acap	4	5	pcon	3
	pcap	4	5	site	4
	apex	4	5	docu	4
	plex	1	4		
	ltex	1	4		
	tool	1	2		
	sced	3	5		
	Ksloc	2.5	165		

Figure 5.2: JPL flight circa 1970 and 1980

ifications that are catered to these flight and ground systems. These are indicated in Figure 3.2. Note that the only difference in the specifications of the JPL flight and ground are in the tactical policy, where ground tactical adds *time* and *stor* in the search space.

Figure 5.5 and Figure 5.6 present the results of the STAR analysis in the form of braille charts. for the columns in these charts, “s” designates use of the strategic policy, while “t” designates use of the tactical policy. In addition, for the projects, “f” and “g” indicate flight and ground respectively, with the numbers following indicating which decade the particular project template

project	ranges			values	
	feature	low	high	feature	setting
JPL ground software circa 2000	sced	3	5	prec	4
				flex	2
				resl	3
				team	4
				pmat	3
				rely	3
				cplx	3
				data	2
				ruse	4.5
				time	3
				stor	3
				pvol	2
				acap	4
				pcap	4
				pcon	4
				apex	5
				plex	5
				ltex	3
				tool	1.5
				site	4.5
				docu	3
				Ksloc	378.42

project	ranges			values	
	feature	low	high	feature	setting
JPL ground software circa 1990	prec	2	5.5		
	flex	2	5		
	resl	2	4		
	team	2	5		
	pmat	2	4		
	cplx	2	5		
	data	2	4		
	ruse	2	4.5		
	time	3	4		
	stor	3	4		
	pvol	2	4		
	acap	3	4.5		
	pcap	3	5		
	pcon	1.5	5		
	plex	2	5		
	tool	1	4.5		
	site	4	5		
	docu	2	5		
	Ksloc	10.5	900		

Figure 5.3: JPL ground circa 1990 and 2000

project	ranges			values	
	feature	low	high	feature	setting
JPL ground software circa 1980	prec	2	6	ruse	3
	flex	2	6	pcon	3
	resl	2	6	site	4
	team	3	6		
	pmat	1	3		
	rely	2	5		
	cplx	2	6		
	pvol	2	4		
	acap	3	5		
	pcap	3	5		
	apex	3	5		
	plex	2	5		
	ltex	3	4		
	tool	1	4		
	sced	2	5		
	docu	2	4		
	Ksloc	0.81	333		

project	ranges			values	
	feature	low	high	feature	setting
JPL ground software circa 1970	prec	2	6	ruse	3
	flex	2	6	pcon	3
	resl	2	6	site	4
	team	3	6		
	pmat	1	3		
	rely	2	5		
	cplx	3	6		
	stor	3	5		
	pvol	2	4		
	acap	3	5		
	pcap	3	5		
	apex	3	5		
	plex	1	4		
	ltex	3	4		
	tool	1	2		
	sced	2	5		
	docu	2	4		
	Ksloc	63	980		

Figure 5.4: JPL ground circa 1970 and 1980

belongs to. Note that any project recommendations produced by STAR are rounded up.

There are several results here that can be observed. One of these is a result that confirms our findings in Chapter 4 concerning the importance of advanced software engineering tools. Here, the recommendation to use $aa = 6$ and/or $ett = 6$ appears $\frac{13}{16}$ times, noting how only the highest values available for these tools are recommended. In addition to this, we can also see how *tool*

project	f00	f70	f80	f90	g00	g70	g80	g90
policies	s t	s t	s t	s t	s t	s t	s t	s t
acap = 4.0	•							•
acap = 5.0	•	•	•	•		•	•	•
apex = 4.0	•	•	•					
apex = 5.0		•		•			•	•
aa = 6.0	•	• •	• •	• •	• •	• •		
cplx = 2.0		• •						
cplx = 3.0							• •	
cplx = 4.0	• •			• •		• •		
cplx = 5.0	• •		• •					
docu = 3.0							• •	•
docu = 4.0				• •				
ett = 6.0		•			• •	•	• •	
flex = 2.0	•							
flex = 3.0	•							
flex = 5.0								•
flex = 6.0				•		•		
ltex = 3.0		•						
ltex = 4.0	•	•	•			•	•	
ltex = 5.0				•				•
pcap = 4.0	•	•	•					
pcap = 5.0		•				•	•	•
pcon = 5.0	•			•				•
peer = 6.0					• •			
plex = 4.0		•	•	•		•		
plex = 5.0	•						•	•
pmat = 3.0			•			•	•	
pmat = 4.0	•			•				•
prec = 3.0	•							
prec = 5.0				•				
prec = 6.0						•		•
pvol = 2.0	•	•						
pvol = 3.0	•	•						
pvol = 4.0		•						

Figure 5.5: Policies produced for the JPL projects, part 1

is recommended across the board at its highest levels available to that particular project template. This results implies that, no matter the time period or the development tools available, project managers should seek to apply these tools at there highest levels available for that project.

Another result that can be observed is in the reductions that the recommendations produced by STAR generate, and how that relates to how constrained projects are. Notice how the most constrained projects (ground 2000, flight 1970 and 1980) have the least reductions. On the other hand, the most unconstrained project (flight 1990) exhibits that largest reductions compared to the initial estimates of that project template. This is a result that is also seen in Chapter 4, as well as presented in a workshop paper by Menzies et. al [58], where it is observed that "if you fix

project	f00		f70		f80		f90		g00		g70		g80		g90	
policies	s	t	s	t	s	t	s	t	s	t	s	t	s	t	s	t
stor = 4.0													•		•	
time = 4.0											•		•		•	
resl = 3.0	•						•									
resl = 4.0	•														•	
resl = 6.0											•		•			
ruse = 2.0	•	•														
ruse = 3.0							•	•								
sced = 2.0	•														•	
sced = 3.0	•		•				•		•		•		•			
sced = 4.0			•		•											
site = 5.0															•	•
site = 6.0	•	•					•	•								
tool = 2.0			•	•	•	•					•	•				
tool = 3.0							•	•					•	•		
tool = 4.0													•	•		
tool = 5.0	•	•													•	•
models	percentage of original value															
effort	51	40	76	52	65	75	25	29	81	78	45	34	45	49	43	58
defects	53	49	88	40	74	53	16	22	29	16	33	15	64	40	40	63
threat	00	00	00	00	00	70	00	00	00	00	00	00	00	00	00	00
months	79	63	89	64	86	85	62	60	91	92	77	71	77	65	74	65

Figure 5.6: Policies produced for the JPL projects, part 2. This figure also presents the differences produced by the policies

everything, you have nothing else to fix".

In the following section, we will present a study similar to this one, however applied to future projects in order to study possible future trends in software engineering. Also, the project templates themselves are meant to be more descriptive of software engineering practices at large and how leading experts see that field progressing in the medium term.

5.2 Futures

5.2.1 Data Collection

In this section we will describe how data was collected in order to provide some insight on future trends in the software industry. That will be followed with presenting the results of this data collection and how those were used in STAR.

The panel meeting was conducted during ICSE 2008 on May 13. Figure 5.7 are photos of this

Figure 5.7: Pictures of the panel at ICSE 2008.

feature	setting	feature	setting
aa	3	tool	3
acap	3	time	3
apex	4.5	team	3.5
cplx	3	stor	3
data	4	site	4.5
docu	3	sced	3
ett	2	ruse	4
flex	3	resl	3
ltex	4.5	rely	2.5
pcap	3	pvol	2
pcon	3	prec	4.5
peer	2.5	pmat	2.5
plex	4.5		

Figure 5.8: The baseline settings representing present day software engineering

panel. It was composed of seven seasoned experts in the field of software engineering:

- Barry Boehm
- Vic Basili
- Ray Madachy
- Thomas Ostrand
- Deiter Rombach
- Rick Selby

	ranges		values	
feature	low	high	feature	setting
aa	3	6	data	4
acap	3	5	docu	3
apex	1	4.5	ltex	4.5
cplx	3	6	pcap	3
ett	2	6	plex	4.5
flex	3	5	team	3.5
pcon	1	3		
peer	2.5	6		
pmat	2.5	5		
pvol	2	5		
rely	2.5	5		
resl	3	5		
ruse	2	6		
sced	1	3		
site	1	4.5		
tool	3	5		

Figure 5.9: Common settings for all future trend variants

- Elaine Weyeuker

These experts formed the part of the panel from whom insights concerning the future trends in the software industry was derived. They discussed projections concerning all the COCOMO II/COQUALMO factors. Note that the time frame for these projections is within a decade. Other assumptions concerning the future projects include the following:

- The projects/applications are *NASA-like* applications.
- They are *mission critical real-time* applications.
- They are *contractor built*.
- They have a *long life time* (between 5 and 10 years).

Three facilitators were present as well:

- Bojan Cukic
- Tim Menzies
- David Raffo

The role of the facilitators was to guide the discussion on the model attribute projections in an attempt to keep the discussions focused. Gregory Gay acted as scribe during this session in order to transcribe notes from the meeting.

During discussions, consensus was rarely reached concerning projections about how the software environment might change relative to the models. For this reason, after brief discussions about every attribute, a vote was taken concerning the future behavior of the attribute. These behaviors include the following:

- **Bump**: The attribute would increase in value at first, and then decrease later on. PREC was voted to have such behavior.
- **Rick**: This is the opposite of bump, where attribute would decrease in value at first, and then increase later on.
- **Up**: The attribute would increase in value. ACAP was voted to have such behavior.
- **Down**: The attribute would decrease in value. APEX was voted to have such behavior.

Some chose to abstain from some of the votes as well. Figure 5.13 and Figure 5.14 contain the summary of the results of these discussions, as well as the votes.

Note that in order to have future projections, we had to set a baseline that represents the current trends in software engineering. This had been set prior to the panel session with input from one of the expert panelists, Barry Boehm, and is presented in Figure 5.8 . From this baseline, we branched 8 main projections. the reason being that, as we see in Figure 5.13 and Figure 5.14 , only 3 attributes (PREC, STOR and TIME) were voted as not having a single direction of variation, namely bumps. Figure 5.9 shows all the other attributes that were voted to have a single direction of change, and there corresponding ranges and values.

5.2.2 Policies

When referring to Figure 5.10, Figure 5.11 and Figure 5.12 , we notice that the policies that are chosen by STAR are very similar across project sizes and projections. This would seem to suggest that, even though we have 8 separate base projections, the overall projection is stable and determinant enough to actually draw stable conclusions from that would apply across the board. In other words, for the purposes of STAR, the experts panel from whom the projections originate don't necessarily need to have consensus for us to draw stable future predictions about policies that need to be applied in the future.

One interesting effect that we see here relates to ASE tools. On the one hand these tools are recommended for the majority of our case studies. This applies most to those cases with full or technical analysis, and to a lesser extent to those with strategic analysis. Note however that in all cases, whenever these tools are applied, they are applied at very high or extremely high levels (between 5 and 6), which emphasizes the importance of advanced ASE tools for future projects. These tools however have the most utility whenever the project is in the development phases. This explains our result since statistical analysis is more related to the planning stages of a project, which tactical analysis is more related to the development stage of a project. Also notice that for the same analysis methods the emphasize the use of advanced ASE tools, the TOOL is recommended at its highest value, which means having it integrated in to the life cycle according to Figure 2.1. So there seems to be a correlation between having the need for ASE tools, and requiring having these tools integrated into the software life cycle.

LARGE
(KLOC = 2500 to 7500)

projection	(p- s- t-)			(p+ s- t-)			(p- s+ t-)			(p- s- t+)			(p+ s+ t-)			(p+ s- t+)			(p- s+ t+)			(p+ s+ t+)		
policies	a	s	t	a	s	t	a	s	t	a	s	t	a	s	t	a	s	t	a	s	t	a	s	t
acap = 4.5										•														
acap = 5	•	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•	
apex = 4				•			•															•		
apex = 4.5	•	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•	
aa = 5.5													•											
aa = 6	•		•	•		•	•	•	•	•	•	•	•	•	•	•		•	•		•	•		•
cplx = 3						•												•			•			
cplx = 3.5				•		•			•	•		•	•									•		•
cplx = 4															•									
ett = 5.5									•															
ett = 6	•	•	•	•	•	•	•		•	•		•	•		•	•		•	•		•	•	•	•
flex = 4.5						•			•									•			•	•		•
flex = 5	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•		•
pcon = 2.5				•																				
pcon = 3	•	•		•	•		•	•		•			•	•		•	•		•			•	•	
peer = 6	•		•	•	•	•	•		•	•		•	•	•	•	•	•	•	•		•			
pmat = 4.5													•						•			•		
pmat = 5	•	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•	
prec = 4																			•					
prec = 4.5	•	•					•	•		•	•		•						•	•		•		
prec = 5				•	•								•	•		•	•					•	•	
pvol = 2.5	•			•			•			•									•					
rely = 4.5																						•		
rely = 5	•			•			•			•			•			•			•			•		
resl = 4.5			•						•				•					•	•		•	•		•
resl = 5	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•		•
ruse = 2																					•			
ruse = 2.5			•			•	•		•	•		•	•		•			•				•		•
ruse = 3									•															
sced = 1.5						•							•		•									
sced = 2			•	•		•	•		•	•		•	•		•	•		•	•		•	•		•
sced = 3			•															•	•					
site = 4.5	•	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•	
stor = 3	•		•	•		•			•	•		•				•		•						
stor = 3.5															•				•			•		•
stor = 4													•								•			
time = 3	•			•			•			•			•			•								
time = 3.5																			•			•		
time = 4										•									•					
tool = 4.5				•		•							•		•			•			•	•		•
tool = 5	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•		•

models	percentage of original value
effort	28 37 54 | 29 49 55 | 24 38 57 | 17 36 56 | 31 40 52 | 25 42 60 | 23 49 53 | 26 39 50
defects	04 22 16 | 05 17 15 | 05 23 18 | 04 35 17 | 07 24 16 | 05 27 17 | 05 47 15 | 10 41 25
threat	00 00 00 | 00 00 00 | 00 00 00 | 00 00 00 | 00 00 00 | 00 00 00 | 00 00 00 | 00 00 00
months	60 69 92 | 62 75 76 | 55 69 77 | 50 68 75 | 62 73 75 | 61 75 85 | 61 74 75 | 59 72 77

Figure 5.10: Future results for large projects

5.2.3 Reductions

The lower parts of Figure 5.10, Figure 5.11 and Figure 5.12 refer to the size of the final model prediction values compared to the initial ones, i.e. the ones before applying any kind of policies

Medium
(KLOC = 250 to 750)

projection	(p- s- t-)			(p+ s- t-)			(p- s+ t-)			(p- s- t+)			(p+ s+ t-)			(p+ s- t+)			(p- s+ t+)			(p+ s+ t+)		
policies	a	s	t	a	s	t	a	s	t	a	s	t	a	s	t	a	s	t	a	s	t	a	s	t
acap = 4.5							•												•			•		
acap = 5	•	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•	
apex = 4							•			•									•					
apex = 4.5	•	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•	
aa = 6	•		•	•	•	•	•		•	•	•	•	•		•	•	•	•	•			•		•
cplx = 3							•			•		•							•					
cplx = 3.5			•	•		•	•		•				•		•	•		•	•		•	•		•
ett = 6	•		•	•	•	•	•		•	•		•	•	•	•	•		•	•					
flex = 4.5							•		•												•	•		•
flex = 5	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•		•
pcon = 2							•																	
pcon = 2.5													•						•			•		
pcon = 3	•	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•	
peer = 6	•		•	•	•	•	•		•	•		•	•		•	•		•	•		•			
pmat = 4.5							•						•						•			•		
pmat = 5	•	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•	
prec = 4							•																	
prec = 4.5	•	•					•	•		•	•		•			•			•	•		•		
prec = 5				•	•								•	•		•	•					•	•	
pvol = 2																						•		
pvol = 2.5	•			•			•			•			•									•		
pvol = 3																•			•					
rely = 4.5																			•			•		
rely = 5	•			•			•			•			•			•			•			•		
resl = 4.5																			•	•		•		•
resl = 5	•		•	•		•	•		•	•		•	•		•	•		•	•	•		•	•	
ruse = 2				•		•										•		•	•			•		•
ruse = 2.5	•						•		•	•		•	•		•	•		•			•			
ruse = 3			•																		•			
sced = 1				•		•										•								
sced = 1.5	•						•						•		•						•	•		•
sced = 2			•				•		•	•		•				•		•	•		•	•		•
site = 4																			•			•		
site = 4.5	•	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•	
stor = 3	•		•	•		•	•			•		•				•		•				•		•
stor = 3.5													•		•						•			
stor = 4																						•		•
time = 3	•			•			•			•			•			•			•			•		
time = 3.5										•						•						•		
time = 4																			•					
tool = 4.5			•						•												•	•		•
tool = 5	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•		•
models	percentage of original value																							
effort	27	42	64	21	46	50	21	46	50	16	40	46	26	41	46	22	48	57	22	43	62	25	44	65
defects	05	36	18	04	10	15	04	10	15	03	24	13	05	39	14	05	28	16	05	39	44	17	45	46
threat	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
months	57	72	81	58	77	78	58	77	78	53	70	77	59	74	75	53	79	57	55	73	79	59	78	79

Figure 5.11: Future results for medium projects

on the case study in question. So smaller is better. We notice four main effects in these tables.

The first effect is that the reductions are very similar across case studies of the same kind of analysis. This doesn't apply in all cases however. The exceptions included such cases where ASE

Small
(KLOC = 25 to 75)

projection	(p- s- t-)			(p+ s- t-)			(p- s+ t-)			(p- s- t+)			(p+ s+ t-)			(p+ s- t+)			(p- s+ t+)			(p+ s+ t+)		
policies	a	s	t	a	s	t	a	s	t	a	s	t	a	s	t	a	s	t	a	s	t	a	s	t
acap = 4.5				•			•	•								•						•		
acap = 5	•	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•	
apex = 4	•						•			•			•	•								•		
apex = 4.5	•	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•	
aa = 6	•		•	•		•	•		•	•		•	•		•	•		•	•					•
cplx = 3	•			•		•	•		•	•		•				•		•				•		•
cplx = 3.5			•	•		•							•		•						•	•		•
cplx = 4									•										•					
ett = 6	•		•	•		•	•		•	•		•	•		•	•		•						
flex = 4															•									
flex = 4.5									•													•		•
flex = 5	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•		•
pcon = 2.5				•						•			•			•						•		
pcon = 3	•	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•	
peer = 6	•		•	•		•		•	•				•			•		•				•		•
pmat = 4.5																			•			•	•	
pmat = 5	•	•		•	•		•	•		•	•		•	•		•	•			•		•	•	
prec = 4										•									•					
prec = 4.5	•	•		•			•	•		•	•		•	•		•			•	•		•		
prec = 5				•	•								•	•		•	•					•	•	
pvol = 2																			•			•		
pvol = 2.5	•			•			•			•			•			•						•		
rely = 4.5																			•			•		
rely = 5	•			•			•			•			•			•			•			•		
resl = 4									•															
resl = 4.5				•	•				•	•	•										•	•		•
resl = 5	•		•	•	•		•		•	•	•		•		•	•	•		•	•		•	•	
ruse = 2										•	•		•				•		•	•		•	•	
ruse = 2.5	•		•	•	•				•	•	•		•		•	•	•				•	•		•
ruse = 3							•																	
sced = 1	•			•	•											•	•							
sced = 1.5	•		•	•	•		•		•	•		•	•		•	•	•		•			•		•
sced = 2									•	•		•	•		•	•	•				•	•		•
site = 4																•								
site = 4.5	•	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•	
stor = 3	•		•	•		•				•		•				•		•				•		•
stor = 3.5									•												•			
stor = 4															•				•					
time = 3	•			•			•			•			•			•						•		
time = 3.5										•						•			•			•		
tool = 4.5	•									•	•		•			•	•					•	•	
tool = 5	•		•	•		•	•		•	•		•	•		•	•		•	•		•	•		•

models	percentage of original value
effort	25 48 68 | 28 49 62 | 31 44 59 | 25 46 66 | 30 52 58 | 30 51 58 | 20 45 48 | 21 48 48
defects	04 36 16 | 05 44 17 | 07 26 17 | 07 36 28 | 06 48 43 | 06 45 16 | 11 40 64 | 14 42 39
threat	00 00 00 | 00 00 00 | 00 00 00 | 00 00 00 | 00 00 00 | 00 00 00 | 00 00 00 | 00 00 00
months	58 75 80 | 62 79 77 | 62 77 80 | 59 76 82 | 64 80 79 | 60 78 74 | 55 77 78 | 57 77 74

Figure 5.12: Future results for small projects

tools aren't applied at all. These tend to have much lower reductions in defects compared to other cases with these tools were being applied. However, the overall effect seems to be that overall the reductions are similar for the same analysis types regardless of the projection or project size. This

is mainly due to the similarity in the policies that are followed, which is mentioned in the previous section.

Secondly, effort reductions seem to be larger for strategic compared to tactical analysis. This would suggest that strategic long term planning is more effective at reducing cost for future projects. Note that our strategic analysis with STAR didn't decisively recommend the use of advanced ASE tools. So there seems to be an argument that ASE tools are necessary for cost reduction. However this isn't the whole picture.

We also observe that defect reductions are largest with tactical analysis. This is due to the affinity of this kind of analysis to recommend ASE tools be used. ASE tools are mainly defect reduction tools, so its only natural that the analysis that most recommends these tools achieves higher defect reductions. The fact that its tactical analysis suggests that ASE tools are best used during the development stages of a software product rather than during the planning stages.

The last effect that we observe is that full analysis achieves the best reductions by a large margin. This applies for all the models and all the case studies, suggesting that if we could control everything in a project, we should. This unfortunately isn't really possible nor feasible. However this can be interpreted as using both proper strategic and tactical planning when appropriate. So, use strategic planning on a future project, apply the policies, then transition to tactical planning and apply the associated policies. By doing this, a software project could achieve a similar effect to controlling all, or most, of the variables in a software project achieving high reductions in effort, defects, and months (schedule). Note that this type of analysis recommends used advanced ASE tools most of the time, providing a strong business case for them.

Note that the threat model doesn't seem to have much importance in our case studies in that model never seem to register any tangible values in any of the experiments. We have seen this before, where for less constrained case studies threat doesn't really come into play (§4.2).

attribute		bumps	up	down	flat	rick	abstain	majority trend	
aa: automated analysis	this one refers to investment in automated analysis tools. Just to clarify,greater investment means that you ”detect and remove” more defects rather than ”gets you more” defects		5		3			*up*	Divided into subsystems. Mission-critical: Up(5), else: level (3)
acap: analyst capability			5		3			*up*	ACAP - analyst capability, problem of people leaving companies early, everyone on Apollo retired. Though, that may relate more to APEX. In some ways, people's capabilities are improving - people are learning things in high school now. Argument that capability goes up, In some experience goes down. Vote: up(5), flat(3)
apex: analyst experience				7	1			*down*	APEX - applications experience. Development platforms and missions continue to change. Systems continue to get more complex . This cannot help but go down over time as NASA engages in new unprecedented missions. Vote: down(7), flat (1)
cplx: product complexity			7					*up*	CPLX - just about any kind of complexity. Advocates for up. Voted: Up (7), flat(1)
data: database size per LOC					4	3	1	*anywhere*	DATA - is ratio of num items in database to loc. Do we need to change scale over time? “growth and data volume matched by technology improvements.” Advocacy for flat and bimodal. Voted: flat (4), rick (3), abstain(1)
docu: required levels of documentation					6		2	*flat*	DOCU - documentation you are required to produce relative to what's needed. Voted: flat (6), abstain(2)
ett: execution-based testing and tools	not a cocomo factor but a factor in the coqualmo defect removal model.		4	1	3			*anywhere*	Some argument that as CPUs get cheaper, we will see more use of CPU-intensive simulation -based methods. Voted: Up(4), down(1), flat(3)
flex: development flexibility	development flexibility. “lo” means development process rigorously. “hi” means only general goals defined		8					*up*	FLEX=5 things less flexible than they were, pushed to a 3, Tom - flexibility will decrease. Voted: up (8)
ltex:language and tool experience	more experience = less development cost			3	5			*flat*	LTEX - language/tool experience. Vote: down (3), flat (5)
pcap: programmer capability			1		7			*flat*	PCAP - programmer capability. Vote: flat (7), up (1)
pcon: personnel continuity (% turnover per year)				6	2			*down*	PCON - personnel continuity, turnover. Vote: down (6), flat (2)
peer: peer reviews	not a cocomo factor but a factor in the coqualmo defect removal model.		7		1			*up*	REVW - Code review. Another way to remove defects. If more people are using them, trend is up. Voted: Up(7), flat(1)
plex: platform experience				2	6			flat	PLEX - platform experience. Vote: flat(6), down(2)

Figure 5.13: Part 1 of the results of the panel discussions.

attribute		bumps	up	down	flat	rick	abstain	majority trend	
pmat: process maturity			7		1			*up*	PMAT - upsies (7), flat(1)
prec: precendentness	have we built this kind of thing before? (higher values means less familiarity)	4	1	2	1			*up/down*	PREC - going to remain fairly stable, might go up in long run. increasing level of uniqueness for some systems, increasing levels of repeated simple components for others. Is there going to be a software crises? Maybe next 20, probably not next 10. Most don't believe we will build extremely unprecedented software, Tom disagrees. Another theory - all are 6s or 2s. Voted: bumbs (4), up (1), down(2), level(1).
pvol: platform volatility			8					*up*	PVOL - platform volatility. Higher volatility (more changes) = up. More and more custom hardware, custom sensors, etc .Vote: Up(8)
rely: required reliability			8					*up*	RELY - downsie = less concerned with reliability, argument for flat (we've launched all sorts of space missions, we'll continue to do same thing), argument for up (demands go up for more reliable systems),. Bojan - things will be flat. Is rely effort into chasing reliability? RELY is actual system reliability. Voted: Up (8)
resl: architectural or risk analysis			7				1	*up*	RESL - upsies (7), abstain (1)
ruse: reuse	is this product being developed the with intent that it is a reusable component?	1	2	2	2		1	*anywhere*	RUSE - I'm investing money to make things reusable. Flat limited to unprecedented-ness, up based on pressure to build for reuse. Cost of building for reuse. Barry presses for bumps. Some projects are so unique that no once can reuse them. I have no idea what's going on here. If desire to do it is high, cost will be high. Definition - putting effort into reuse. Voted: bump(1), abstain(1), down (2), up(2), flat (2)
sced: dictated development schedule	lower values means developers forced to deliver faster than they would like			7	1			*down*	SCED - schedule pressure, low=fast. Vote: down (7), flat(1)
site: multi-site development	"lo" means some contact: phone, mail; "hi" means multi-media			7	1			*down*	SITE - how spread out people are. Physical location % communication. More people working at home. Vote: down(7), flat(1)
stor: required % of available RAM		7					1	*up/down*	STOR - see above, Voted: bumps (7), abstain(1)
team: team cohesion	lower values means more difficult interactions (e.g. sub-contractors working behind separate firewalls)				8			*flat*	TEAM - level (8)
time: % of CPU		6	1		1			*up/down*	TIME - has been going down, used to be constrained all the time. Resources fixed once something is launched. Voted: bumps (6), up(1), flat(1)
tool: use of software tools	"lo" means just simple edit/debug cycles; "hi" means tools tightly integrated to the development cycle		6		2			*up*	TOOL - how good are tools? Ability to improve productivity. More use of tools like stateflow and auto code generation. up(6), flat (2)

Figure 5.14: Part 2 of the results of the panel discussions.

small	medium	large	
8	8	8	Acap=5
8	8	8	Apex=4.5
8	8	8	Flex=5
8	8	8	Pcon=3
8	8	8	Pmat=4.5, 5
8	8	8	Prec=4.5, 5
8	8	8	Rely=5
8	8	8	Resl=5
8	8	8	Site=4.5
8	8	8	Time=3, 3.5, 4
8	8	8	Tool=5
8	8	7	Sced=1, 1.5, 2
6	7	7	Stor=3, 3.5, 4
8	8	5	Pvol=2, 2.5, 3
8	8	4	Ruse=2, 2.5, 3
8	7	4	Cplx=3, 3.5, 4
7	8	8	Aa=6
6	7	8	Ett=6
5	7	7	Peer=6

Figure 5.15: Future recommendations summary with ALL policy

small	medium	large	
8	8	8	Acap=5
8	8	8	Apex=4.5
8	8	8	Pmat=4.5, 5
8	8	8	Prec=4.5, 5
8	8	8	Site=4.5
8	8	6	Pcon=3
0	3	3	Aa=6
0	2	3	Ett=6
1	1	3	Peer=6

Figure 5.16: Future recommendations summary with Strategic policy

small	medium	large	
8	8	8	Flex=5
8	8	8	Resl=5
8	8	8	Ruse=2, 2.5, 3
8	8	8	Sced=1, 1.5, 2
8	7	8	Stor=3, 3.5, 4
8	8	8	Tool=5
8	8	7	Cplx=3, 3.5, 4
7	7	8	Aa=6
6	6	8	Ett=6
5	7	7	Peer=6

Figure 5.17: Future recommendations summary with Tactical policy

Chapter 6

S-cost and Drastic Control

In this chapter, we will discuss a scheme used by Boehm et al. [16] for conflict resolution in project requirements. This is followed by study that investigates the effectiveness of this scheme compared to just using default analysis within STAR. Note that this study was redone with SEESAW (subsubsection 2.3.2) in a paper presented at **ICSE'09**: *How to avoid drastic software process change*. [64]

6.1 Conflict Analysis with S-Cost

Boehm et al. [16] describe a system called S-Cost. The purpose of this system to semi-automate the process of finding conflicts in cost - quality requirements of software systems. This system also works to facilitate the process of resolving those conflicts by presenting stakeholders in the software project with several strategies for the purpose of finding a proper compromise for all parties. In this section, we will present this system, describing briefly how it operates and explaining the strategies that it presents to stakeholders.

6.1.1 The problem

Quality in software products was being attributed to service oriented customer satisfaction, which caused some qualities to be neglected (e.g. maintainability) and placed too much emphasis on usability. To remedy this, this was expanded to include the evaluation for not only the customer, but also the all the other stakeholders in a software product. This allowed considering cost, maintainability, interoperability and other software qualities as parts of what describe software quality. However, this also introduced the problem of coordinating all the stakeholders needs, causing cost-quality conflicts to emerge. This is exasperated by the complex dependencies that software cost has with the software quality attributes, as well as the large resolution space that is available to find solutions to these cost-quality conflicts. S-Cost was introduced as a semi-automated system in order to assist the stakeholders in visualizing and resolving conflicts by applying certain strategies to different modules of the software product.

6.1.2 S-Cost

S-Cost combines two separate systems to provide its semi-automated risk resolution capabilities: USC-CSE WinWin system and the Qarcc tool. WinWin is based on a spiral model that aims at resolving conflicts between stakeholder needs, otherwise called as “win conditions”, and making everyone a “winner”. This system is manually manipulated, making it hard on stakeholders in larger products to identify the conflicts in their win conditions. Qarcc is a knowledge-based tool that identifies conflicts in software quality requirements, by using predefined stakeholder/quality-attribute relationships, and notifies stakeholders of these possible conflicts. Neither of these systems has the ability to get the effect of conflict resolution on development costs. S-Cost adds this ability, combining both systems to be able to find win-win solutions to quality conflicts, all while using COCOMO (COnstructive COst MOdel) to indicate cost estimates. This allows S-Cost to help resolve cost-quality conflicts as well as quality conflicts between different stakeholders. S-

Cost is able to give suggestions (or strategies) for the stakeholder to apply to resolve cost conflicts, while keeping keeping affected stakeholders of the implications of these resolutions. Visualization and negotiation aids are provided as well by S-Cost.

6.1.3 The strategies

There are several strategies that S-Cost presents to the stakeholder, giving the stakeholder the option of choosing the degree to which to apply it to the different modules of the software product. Each strategy influences a subset of the COCOMO (in this case COCOMO II) software cost attributes, and each has its own pros and cons that are presented to the stakeholder. These strategies are the following:

- **Reduce/defer Functionality:** *KDSI, DATA* - Reducing the functionality of the product will reduce size, hence reducing cost and easing maintenance. This however reduces the abilities of the product which influences customer satisfaction.

- **Reduce/defer Quality:** *RELY, DOCU, TIME, CPLX* - Reducing quality will drive costs and development time down, but will increase maintenance costs especially for products that will have longer life cycles.

Drastic change	**Possible undesirable impact**
1 Improve personnel	Firing and re-hiring personnel leading to wide-spread union unrest.
2 Improve tools, techniques, or development platform	Changing operating systems, IDEs, coding languages
3 Improve precedentness / development flexibility	Changing the goals of the project and the development method.
4 Increase architectural analysis / risk resolution	Far more elaborate early life cycle analysis.
5 Relax schedule	Delivering the system later.
6 Improve process maturity	May be expensive in the short term.
7 Reduce functionality	Delivering less than expected.
8 Improve the team	Requires effort on team building.
9 Reduce quality	Less user approval, smaller market.

Figure 6.1: Nine drastic changes from [10].

- **Improve tools, techniques or platform:** *TIME, STOR, PVOL, TOOL, SITE* - Improving tools and the platform will cause training and platform costs to increase, but will improve maintainability and will reduce software cost and schedule in the long run.

- **Relax the delivery schedule constraint:** *SCED* - Tightening the delivery schedule will allow development time and cost to decrease but may negatively influence product capabilities.

- **Improve personnel capabilities:** *ACAP, PCAP, PCON, AEXP, PEXP, LTEX, KSLOC/PM* - Improving capabilities of analysts and programmers and transferring more experienced ones to the project would reduce the cost of the product while improving its quality. This would mean however that other products will lose skilled personnel, negatively influncing them, and the cost per person will increase due to using more skilled personnel.

- **Reuse software assets:** *ADSI, DM, CM, IM* - Reusing software code will allow the cost of the development to decrease, however the effect on the product's quality depends solely on the quality of the software being reused.

- **Improve coordination via team building:** *TEAM* - Team building allows for removing interpersonal overhead, which drives down the schedule and the cost. This factor isn't always controllable however.

- **Architecture and risk resolution:** *RESL* - Reducing risk resolution will allow more resources to be allocated for product development, reducing the development time and cost. This will cause a need for additional risk management overhead however.

- **Improve process maturity level:** *PMAT* - Increasing this will help with reducing the time and cost of resolving product bugs and hence improves quality. Overhead is introduced with this however due to the need to implement the Software Engineering Institute's Capability Maturity Model.

- **Improve precedentedness and development flexibility:** *PREC, FLEX* - Having increased familiarity with the product type and more flexible development would reduce costs. Such factors are uncontrollable sometimes and depend on the development environment and history.

- **Increase budget:** *Revised win condition* - If Funds are available, this could allow the product to be more competitive and may increase ROI.

6.2 Drastic vs. Conservative Control (Hoh In strategies)

Earlier we presented a paper by Boehm that suggested using strategies in order to impose control over a project and resolve conflicts in project requirements. These strategies were presented earlier, and are meant to override internal project parameters and policies. For this reason we have placed these strategies under the guise of drastic change. In the software world, these drastic solutions to conflicts within projects are suggested that would theoretically modify a project in the intended manner. However these drastic solutions can have unintended effects , where a strategy like reduce functionality might deliver the effect of reduce cost and time of development, but also put in the risk of under delivering. This would not be helpful for the project as that might cause it to be poorly received.

Conservative project changes are changes with in the project that abide by present project parameters, which is the way that star operates by default. We wish to present the following argument: Before performing *drastic changes* to a project, it is worthwhile to thoroughly explore the available options within the current structure of a project. Through applying policies that avoid drastic change, we wish to see whether we are able to achieve the same or better effects compared to drastic changes, while at the same time attempting to avoid the negative side effects that drastic changes inherently have.

The Figure 6.2 presents the various strategies presented by Boehm et al., and there implemen-

Drastic change	Effects on Figure 3.1
1 Improve personnel	acap = 5; pcap = 5; pcon = 5 apex = 5 ; plex = 5 ; ltex = 5
2 Improve tools, techniques, or development platform	time = 3; stor = 3 pvol = 2; tool = 5 site = 6
3 Improve precedentness / development flexibility	prec = 5; flex = 5
4 Increase architectural analysis / risk resolution	resl = 5
5 Relax schedule	sced = 5
6 Improve process maturity	pmat = 5
7 Reduce functionality	data = 2; kloc * 0.5
8 Improve the team	team = 5
9 Reduce quality	rely = 1 ; docu = 1 time = 3 ; cplx = 1

Figure 6.2: Implementing drastic changes.

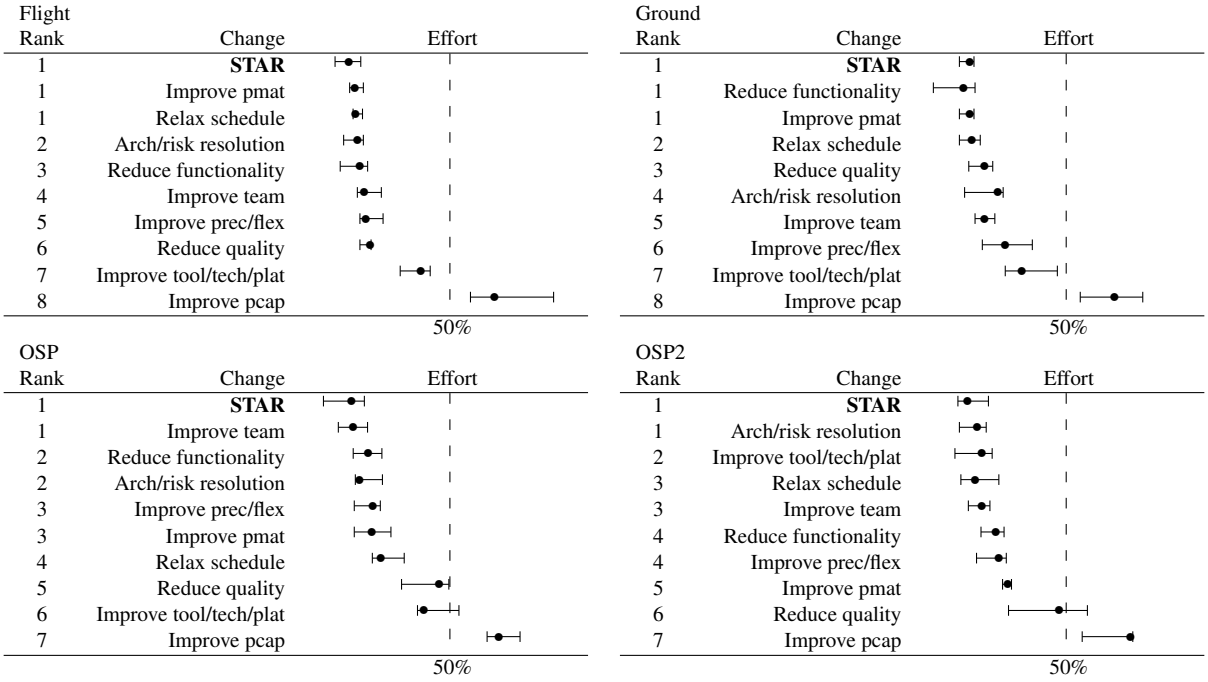


Figure 6.3: Drastic vs. Conservative change; EFFORT: total staff months (normalized 0..100%) with default analysis.

tation within STAR. Note that the setting on the right hand side overrides any project settings that conflict with them. All the numbers indicate ratings, with KSLOC being the only exception, where the number is a ratio that is multiplied by the range that the project has in for the KSLOC attribute.

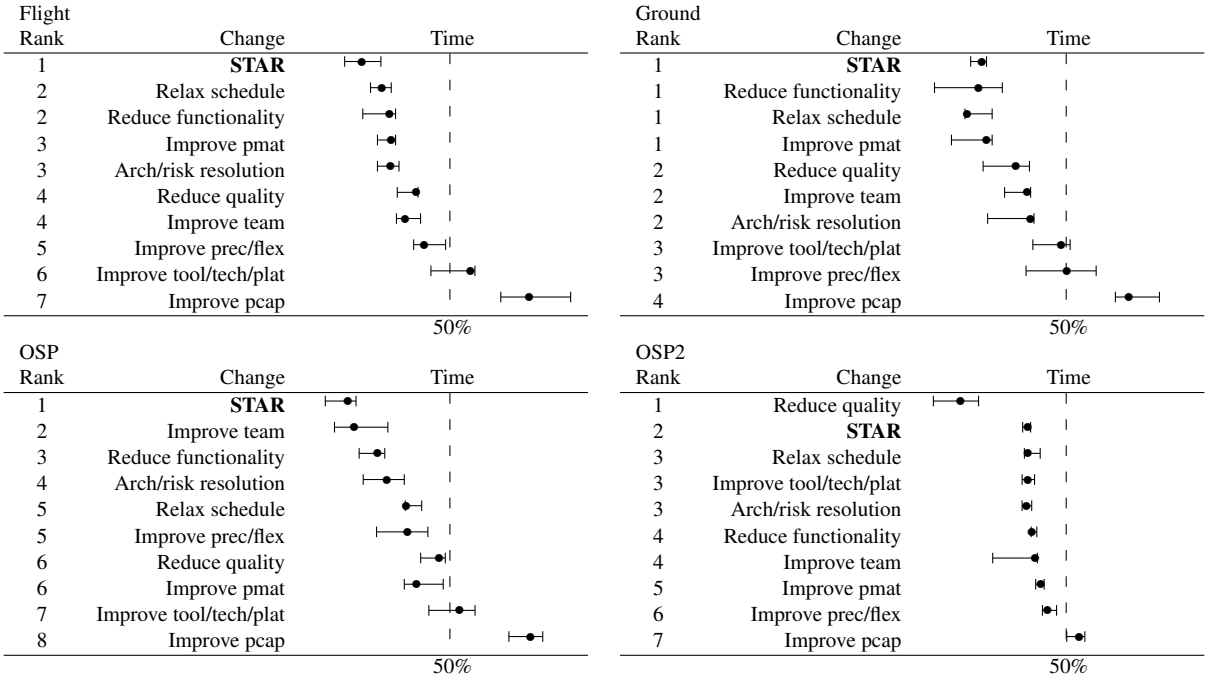


Figure 6.4: Drastic vs. Conservative change; TIME: calendar months (normalized 0..100%).

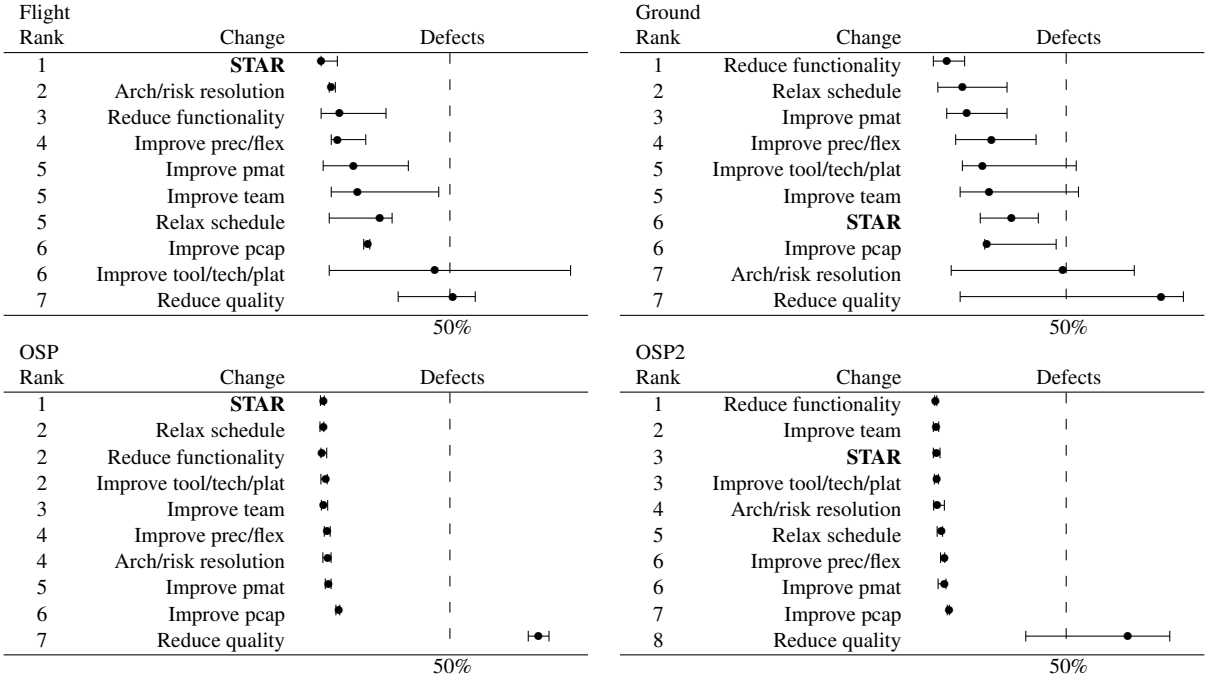


Figure 6.5: Drastic vs. Conservative change; Defects (normalized 0..100%).

As we did in the previous section, this section will also only concentrate on analysing the results for three of the four software engineering models used within STAR. The experiments for each scenario were run ten times, after which the data generated at the policy points was collected and passed through the Mann-Whitney unit test [56], where each strategy is ranked. The total number of runs of STAR in this case study is

$$strategies * projects * 10$$

This calculates to a total of 400 runs, since we used the NASA projects. Note that the above figures are quartile charts that indicate the variations in the runs. Only the second and third quartile are shown in order to reduce the clutter of the graphs that would've been caused by outliers. "**STAR**" indicates conservative change, with none of the strategies being used. Also, the number on the left of the quartile indicate the rank according to Mann-Whitney. Threats weren't included in this study due to results that had it zeroed out across the board, with occasional outliers which produced messy, inconclusive results, a pattern seen several times throughout this thesis.

In terms of the results for effort, clearly avoiding any drastic change within the project is the best choice, ranking first in all the tests. And while it did tie with other strategies for the different projects, conservative change maintained its rank across the multiple projects, irrespective of how constrained a project is. Another clear result is that improving programmer capability (pcap) is definitely not the way to go to maintain costs, ranking last for every project. Other notable observations include the reducing functionality does well until you hit the most constrained projects and that improving the tools used and the platform does best in the most constrained project. Note here however that, despite of the ranking, most of the alternative strategies are within ballpark figures of each other, except for the lowest ranking strategies, making those the most definitive result.

Moving on to development time, there seems to be a similar picture compared to the results under effort. Improving *pcap* is still the worst thing you can do, and avoiding drastic change is still a favorable policy to go by. At worst, conservative change manages to be second for the most

constrained project. On the other hand reducing functionality and relaxing the schedule both do very well with the lightly constrained projects (flight and ground), and improving the tools and platform used does best in the most constrained project. One very interesting definitive result comes from reduce quality in OSP2, where this strategy wins out by a large margin, while it scored mid-pack in all the other projects. This could be interpreted as saying the following: earlier in the project, reducing the quality of the product doesn't help with development time; however, as the development of the product nears the end stages, if you need to rush the project out, reducing the quality (which happens to reduce reliability and documentation) will help in getting the product out the door. A word of caution here however: this short term payoff only comes at a cost in the medium term. As can be seen in the OSP2 results for effort in Figure 6.3 and defects in Figure 6.5, the side effect of rushing a product out is larger effort and much larger defect numbers.

Finally, when it comes to defects, we see a consistent overall result that reducing quality will in fact increase defects. This is demonstrated by having *reduce quality* rank last in all the projects, with an especially large margin in the more constrained project (OSP, OSP2). Here as well though, conservative change does well. It scores lowest with ground; however, looking at the quartile charts for ground, the difference between STAR and the higher ranked drastic change strategies isn't significant enough to declare a clear loss.

To conclude this section, we have and demonstrated that, overall, conservative change and policies that abide by project limits are a better, more consistent choice for project managers. Even though drastic change does seem to do better in some cases, the difference is either not significant enough or the strategy itself has serious side effects on other aspects of the project.

Chapter 7

Better, Faster, Cheaper

In this chapter, we will present the concept of "better, faster, cheaper" (BFC) and its history, followed by a study conducted on BFC.

7.1 History of BFC

"Faster, Better, Cheaper" or "Better, Faster, Cheaper" (BFC) was a philosophy in designing and implementing projects that was pushed in NASA by Daniel Goldin, who served as the NASA Administrator from April 1^{st}, 1992 to November 17^{th}, 2001. It was mainly pushed to reduce the expenditure of NASA, and was in-line with the direction that the Clinton administration's approach of doing more for less. While starting out on the right foot with such successes as Pathfinder, later failures caused a wealth of criticism of these policies. While BFC hasn't been abandoned, its application has since then evolved to rectify its implementation shortcomings.

7.1.1 The Start and Successes of BFC

The view of pre-Goldin BFC was that at one time in a project, only two of the aspects of BFC could be achieved. This was expressed by the common phrase "Better, Faster, Cheaper: pick any

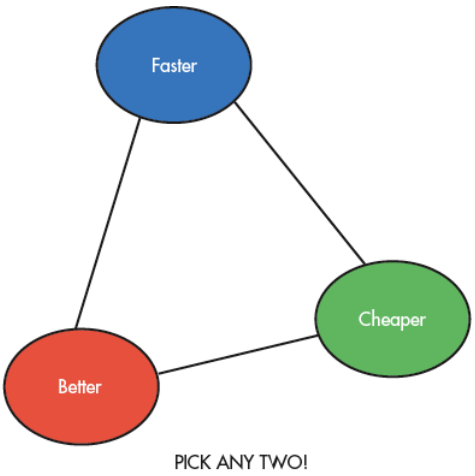


Figure 7.1: Better, Faster, Cheaper [77].

two", illustrated in Figure 7.1. Goldin believed that it was in fact possible to have all three aspects, and so initiated new policies within NASA based on his views.

The main approach to implementing BFC within NASA was to downsize projects and reduce their cost and complexity, concentrating on producing missions in volume. Reducing funding naturally meant that less verification and testing was possible within budget and schedule constraints. The reasoning behind this however was to be able to produce a larger volume of unmanned missions, which would counteract the expected higher rate of mission failure. This would, optimally, yield more successful missions as well as more scientific data produced by these projects. Another focus in this policy was allowing teams to take acceptable risks in projects to allow for cost reduction, and possibly using new technology that could reduce cost while possibly providing more capabilities. This was accompanied by the the new view that was being pushed at NASA by Goldin that "it's ok to fail" [75], which was rather misunderstood. This new policy was meant to eliminate huge budget missions of the past, that upon possible failure would yield large losses. Project cost used to routinely exceed the $1 billion mark, while the first BFC project, the Mars Pathfinder, was

completed for a fraction of the cost, netting at about $270 million [27].

Some within NASA, such as 30 year veteran Frank Hoban, supported these policies [27]. Some viewed these new policies as a necessary break from traditional policies that were very risk averse. The additional cost reduction, accompanied by the additional risk, was to allow for a path to cheap and commercial space flight. Even given the reduced funding, the Mars Pathfinder mission, along with other first generation BFC missions, were successes. This fueled enthusiasm to apply BFC across all of NASA and further reduce spending per mission, as well as reduce NASA expenditure by reducing the workforce by a third. BFC was extended to be applied on manned space missions as well, where funding was also reduced. Coming into a space shuttle program that was starting to age and in need of updates, the new policies imposed cuts in funding from 48% of the NASA budget to 38% [50], further straining that program. Further more, a single prime contractor (Lockhead Martin) was used for missions in another bid to reduce cost and managerial complexity [69, 80].

This produced opposition within NASA, where traditionally issues pertaining to the shuttle were designated LOVC (Loss of Vehicle and Crew) and given priority over all other issues, including cost. However the cost cuts and layoffs that ensued were too much for teams, and caused a blow to morale. In addition there was a progressive loss of veteran scientists, engineers and managers who had accepted offers for early retirement that were extended to them [50].

Despite this, additional projects were on the way in the form of the Mars Climate Orbiter and the Mars Polar Lander. These two projects were more aggressive implementations of BFC, especially when it came to the Faster-Cheaper part of those policies. Costs of the Orbiter and the Lander were brought down to $125 million and $165 million respectively [77]. This was much lower that the previous Pathfinder mission, which itself cost slightly less than $300 million. The success of these missions would've furthered the BFC mantra within NASA and JPL, and would've been seen as breaking new ground in terms of mission completion with the kind of staff and cost reductions they had compared to previous missions, even the Pathfinder [38].

7.1.2 The Legacy of BFC

Given its early success in terms of mission delivery, BFC started being more aggressively applied to missions in NASA. One product of this were the above mentioned Mars Climate Orbiter and Polar Lander. Each cost about 40% less than the previous Pathfinder mission, which is extraordinary given that Pathfinder had been touted as a money saver by NASA when compared to previous missions like Viking, which cost about $935 million in 1974 Dollars (equivalent to $3.5 billion in 1997 dollars). Both of these missions however failed. Using a single contractor had weakened quality assurance and resulted in flaws that caused the loss of these two Mars missions. These flaws had been software flaws that could have easily been rectified if they had been discovered on the ground. One of these flaws was a failure to convert from imperial to metric units, causing the loss of the Climate Orbiter [68]. The Mars Program Independent Assessment Team Report [80] found that these missions were understaffed, underfunded by at least 30%, and too tightly scheduled.

Elsewhere, across the Atlantic in the UK, another mars mission to deliver a lander, designated the Beagle 2, was under way. This mission way also developed cheaply, applying the same concepts in design and implementation that NASA was at the time using. The lander however was declared lost after not establishing contact after separation from the mars express vehicle [1].

One other failure that BFC was blamed for was the Columbia Shuttle disaster in 2003. This was post-Goldin, at a point where NASA had realized the excessive cost cutting and staff reducing policies needed to be changed. After that disaster, critics quickly pointed the finger to these missions being under funded due to BFC. There were many calls, especially politically, for throwing BFC "in the waste basket" [25]. One report that particularly criticized BFC, and Goldin's policies in implementing it, is the International Federation of Professional and Technical Engineers' (IFPTE) report on the Effectiveness of NASA's Workforce & Contractor Policies [73], which stated the following:

> "Using FBC as a way to contract out services and move more of NASA's resources into

> the private sector, Goldin eliminated much of the civil service infrastructure that monitored and held technical knowledge of the service and products contractors provided and oversaw NASA's safe and successful operation. Critics of FBC always doubted NASA's ability to fulfill FBC without sacrificing either the 'faster', the 'better', or the 'cheaper'. Concerns became widespread after the highly publicized Mars missions failed in 1999. Further concerns arose as NASA's workforce reductions and increased contractor workforce, jeopardized the safety of space shuttle operations. Enough evidence existed in failed missions, close calls, and government reports that suggested the trade offs of FBC were inexperienced and reduced workforce capability; increased safety risks; and minor oversights that resulted in lost spacecraft."

Despite all the criticism of these policies, it should be noted that BFC was successful/partially successful in 136 of the total of 146 missions launched during the period that Goldin was administrator. This would be called an overall success if it hadn't been for the largely publicized failure, as it had been suggested that NASA could've used better PR [78]. Also, the major failures in missions attributed to the BFC policies seemed to be due to communication and managerial failures, rather than the BFC policies directly. This can be seen as a result of how BFC was implemented, where staff reductions, as well as the aforementioned loss in veteran engineers and managers to retirement, causing experienced managerial staff to be stretched too thin given tight scheduling [80]. This forced projects to use inexperienced managers which caused management mix ups and human error.

It seamed that BFC was more of a front for faster and cheaper, as the extreme cost cutting and tight scheduling seemed to eat into the quality, i.e. the better, of the missions. This implies that BFC was not the issue in itself, but rather its implementation. In support of this, Tony Spear, a JPL veteran engineer from 1962 to 1998, testified [76] to the possible effectiveness of BFC, and emphasized the importance of a cohesive and communicative team for it to work. He also mentions a fixation that happened in NASA on cost, causing cost cuts that were too much for

2^{nd} generation BFC projects. However, far from recommending throwing BFC out the window, he recommended a more focused way of implementing it, concentrating on three aspects: people, technology, and methods [75,76]. This would entail more staffing and building and retaining talent, taking advantage of advancements in technology such as the Internet, and advancing methods used in project development and verification. One of the recommended methods include peer review, as well as also developing multi-mission technology. This would make the projects and the process by which these projects are developed more mature. Spear elaborated on this in the NASA FBC Task Final Report [75].

A more detailed time line of BFC and its woes within NASA can be seen online [69].

7.2 Analysis of BFC: Pick Any Two?

Having presented the history of BFC at NASA, our purpose behind this section is to attempt to investigate the viability of achieving a project that has high reliability and quality, was cheap to develop and was quick to get to market. The general consensus before the Goldin era at NASA was that you had to have a compromise somewhere. Hence the common phrase "Better, Faster, Cheaper: pick any two".

To apply study this, we needed to apply the different scenarios of BFC within STAR. These include BFC, FC, BF and BC, where the latter three options are the "pick any two" variants. For this study, BFC was interpreted as an indication of what models the weighting is on in Equation 3.2. More specifically, it was interpreted as the following:

- BFC: maintain default weights in the fitness function: $\alpha = 1$, $\beta = 1$, $\gamma = 1$, $\delta = 1$ and $relydefect = 1.8$.

- FC: Apply higher weight on the effort and development time aspects in the fitness function, and eliminate all the others: $\alpha = 2$, $\beta = 0$, $\gamma = 0$, $\delta = 2$ and $relydefect = 0$.

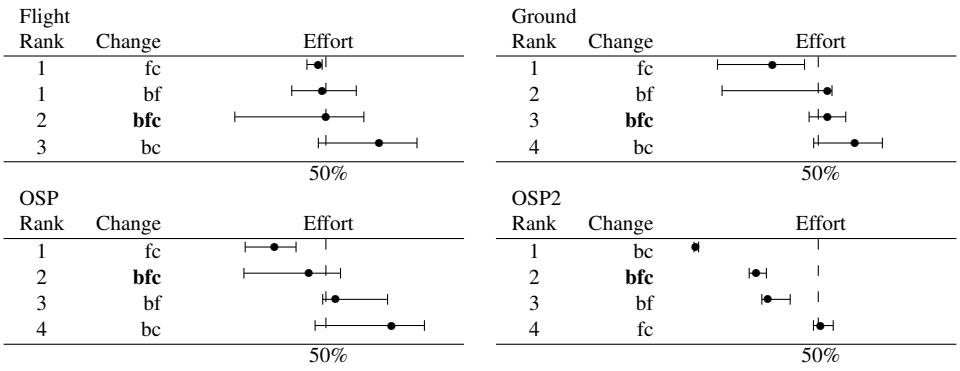


Figure 7.2: Better, Faster, Cheaper; EFFORT: total staff months (normalized 0..100%).

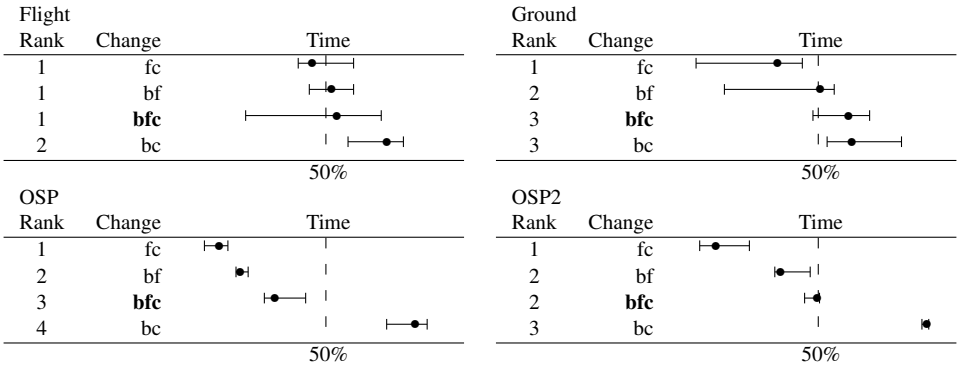


Figure 7.3: Better, Faster, Cheaper; TIME: calendar months (normalized 0..100%).

- BF: Apply higher weight on the defects and development time aspects in the fitness function, and eliminate all the others: $\alpha = 0$, $\beta = 2$, $\gamma = 0$, $\delta = 2$ and $relydefect = 1.8$.

- BC: Apply higher weight on the defects and effort aspects in the fitness function, and eliminate all the others: $\alpha = 2$, $\beta = 2$, $\gamma = 0$, $\delta = 0$ and $relydefect = 1.8$.

For each of the NASA projects presented in §3.4, STAR was run ten times for each of the variant weighting schemes, all while employing default analysis. The results from these runs were post processed to produced the quartile charts presented in Figure 7.2, Figure 7.3, and Figure 7.4. Note that these charts only show the middle two quartiles in an effort to keep the charts legible, and in order to avoid reporting outliers. These quartiles have also been normalized to further emphasize the differences between the different schemes. The rank field in those charts indicates

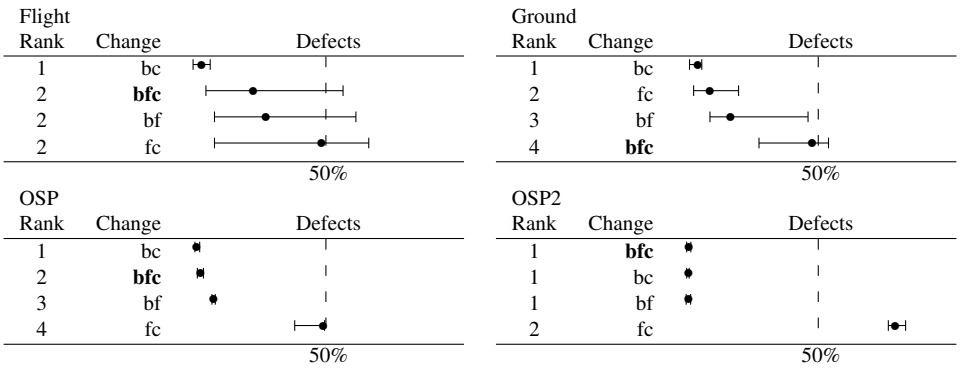


Figure 7.4: Better, Faster, Cheaper; Defects (normalized 0..100%).

the rank given to the particular weighting scheme when statistically compared to the other schemes using the Mann-Whitney unit test [56]. Threats weren't included in this study due to results that had it zeroed out accross the board, with occational outliers which produced messy, inconclusive results, a pattern seen several times throughout this thesis.

Looking at Figure 7.2, the figure describing effort results, we observe that BFC seems to perform mid-pack compared to the rest of the weighting schemes, while FC ranks first in all but one project (OSP2). For OSP2, BC ranks first among the weightings, with FC ranking last. Note that in all the other projects, BC actually ranked last. However, those rankings can't be considered a clear loss for BC since the range of effort estimates produced by BC overlaps fairly well with the estimates from the other weighting schemes. FC's loss in OSP2 however is a more clear loss.

Moving on to Figure 7.3, the figure describing development time results, we can again see how BFC performs mid-pack compared to the other schemes. This time around, FC wins across the board, with BF trailing close behind, and BC rounding out the pack. The more decisive wins for FC take place with the more constrained projects, OSP and OSP2, as with the more decisive losses of BC.

Finally, in Figure 7.4, the figure describing defects results, we observe that BFC does very well with OSP and OSP2, performing on par with BC and BF for these projects. On the other hand, FC does very poorly for these projects, clearly driving defects up due to a lack of emphasis on defects.

The performance in the other two projects is closer, however note that BFC does slightly worse than everything else with *ground*.

From the above observations, we present the following analysis relating to BFC:

1. BFC is a "jack of all trades, master of none". It works well in most instances, and provides a compromise solution in balancing out model estimates. This supports the view that you can't fully achieve BFC with out some compromises, and that you're better off "picking any two" aspects to concentrated if there is a focused end result is needed.

2. FC speeds projects, while having the side effect of drastically increasing defects for more defined and constrained projects. This seems to suggest against rushing well defined and constrained projects out the door, and concentrating on on keeping some "better" aspect in consideration to avoid the defect spike, especially for mission critical projects.

3. Finally that BC, while driving down defects, also drives up development time, especially for more constrained projects. This also applies for effort for all projects other than the most constrained (OSP2)

So as a final conclusion to this section, this study seems to support "better, faster, cheaper: pick any two", providing an empirical study that opposes the BFC policies applied throughout NASA in the 1990's to the early 2000's.

Chapter 8

Conclusions

Through the course of this work, we have presented a tool that uses AI search in order to produce project recommendations and estimates. The two major conclusions that we are able to draw from the empirical studies that preceeded are:

1. Lack of data, whether for present or for future projects, is not a show-stopper in terms of our ability to explore possible policies to use. Rather, this can be exploited in order to produce recommendations that can reduce estimates substantially (see Figure 4.18, Figure 5.10, Figure 5.11, and Figure 5.12).

2. With a such a system, methods used in software engineering and development (like those presented in Chapter 6 and Chapter 7) can be evaluted and tested. Doing so, we found that drastic change within a project is not advisable, and that optimizing for all models at once will yeild some compromises.

As for the summary of the empirical results that were presented in the seperate chapters:

- Chapter 3: Within the space of *Project* options that most reduce *Estimation* median and variance, the predictions made by our process models are remarkably similar to those made by conventional methods (see Figure 3.8 and Figure 3.7).

- Chapter 4: *Estimation* median values can be greatly reduced (see Figure 4.18). In comparisons with other effort estimation tools, the reduction can be quite dramatic. In the best case our tools found *Project* ranges that yields estimates that were 5% of estimates found by other means. In addition, *Estimation* variance can be reduced by only varying the *Project* values and leaving the *Model* values free (see Figure 4.2 to Figure 4.16). Also, varying a small amount of model values makes a big difference. Also, advanced *Automated Software Engineering* tools are essential for software projects, and become more necessary as projects mature and near the end of their development cycle. They are also necessary for future projects (see §4.2). These tools are also only recommended to be applied at there highest values.

- Chapter 5: Two case studies are presented. For the first JPL case study, several results were observed. One result was that it confirmed our findings in Chapter 4 concerning the importance of ASE tools. Another result was that the most constrained projects (ground 2000, flight 1970 and 1980) had the least reductions, while the most unconstrained project (flight 1990) exhibited the largest reductions compared to the initial estimates of that project template. This is a result that is also seen in Chapter 4, as well as presented in a workshop paper by Menzies et. al [58], where it is observed that "if you fix everything, you have nothing else to fix".

- Chapter 6: The study here showed that conservative change and policies that abide by project limits were a better, more consistent choice for project managers. Even though drastic change did seem to do better in some cases, the difference was either not significant enough or the strategy itself had serious side effects on other aspects of the project.

- Chapter 7: BFC is a jack of all trades, achieving reductions accross all the models with moderation compared to the focused "pick any two" practices like FC for example for speeding project delivery.

Finally, we comment on the external validity of these results. Compared to many other process models[1] this combination of effort/schedule/threat/defect models is relatively simple. As model complexity grows, then the space of possible *Estimates* can grow exponentially and STAR's controlling effect may disappear. Therefore it is clear that we can not claim that, for *all* process models, that *Estimate* variance can be controlled by just constraining *Project*, not *Model*, variance.

Nevertheless, data collection for the purposes of model calibration remains as a expensive, tedious, and often incomplete process. Our results suggest that such data collection may be, for some process models, an optional activity (caveat: provided that a process model exists that specifics the general relationships between concepts in a domain). In this work, we managed to establish the following strategy:

- finding the fewest number of variables that most effect model output;
- constrain them;
- check for stable conclusions in the constrained space.

If these results from STAR generalize to more complex models, then is should be possible to make reasonably accurate predictions without local calibration data.

Note that if such stability is *absent* in more complex models, and those models are being used in domains with data collection problems, then we would argue that that is a reason to abstain from such complexity, and use COCOMO-style models instead.

8.1 Future work

This work has been used as a proof of concept for further work that is currently being undertaken by several other graduate students at the WVU LCSEE dept. These include the following:

[1] See Software Process journal, issue on Software Process Simulation, vol. 7, No. 3-4, 2002.

- In this work, we use simulated annealing. In other work [64] williams explores another stochastic search engine, SEESAW, to compare the recommendations of his version of STAR to DELPHI recommendations by experts.

- Green [34] is using our design to assess the impact of value-based software engineering on standard process control based on [46].

- Orrego [71] is using our design to assess the value of reuse within the context of reducing effort, defects, schedule, and threats.

Aside from the above, other future work include deriving several other publications from this work.

Appendix A

Obtaining and Using STAR

These instructions should support a LINUX and CYGWIN install of STAR 2.1. In the event of technical difficulties, please contact the author.

```
bash -i
[ ! -d "$HOME/bin" ] && mkdir $HOME/bin
export PATH="$PATH:$HOME/bin"
wget http://unbox.org/wisp/tags/STAR/2.1/STAR_2.1.zip
unzip STAR_2.1.zip
make  # Requires gcc
cd ~/STAR
cd eg  # Accessing the folder with all the example scripts
STAR -h   # Showing the STAR help file
```

To download and run the trunk version of STAR of the SVN repositories, do the following in place of downloading off of the site directly.

```
svn checkout http://unbox.org/wisp/trunk/STAR/ # Requires Subversion to be installed
make
```

Appendix B

Source Code

B.1 main.cpp

```
#include "project.h"
#include "minmax.h"
#include "randomNum.h"
#include <iostream>
using namespace std;
#include <cstdlib>
#include <cmath>
#include <ctime>
#include <iomanip>
#include <cstring>
#include <sstream>
#include <fstream>
#include <stdlib.h>


double energy(project);
double energyM(project);
double calcEnergy(project, double, double, double, double);
double prob(double, double, double);
project neighbour(project, double[], int, bool);
double temp(int, int);
void outStats(project);
void partition(double[], int, int, int&, int&);
void quicksort(double[], int, int);
void sort(double[], int);
void commandLine(int, char*[], double*, double*, int*, bool*, bool*, bool*, bool*, bool*, bool*, project*,
bool*, bool*, bool*, bool*, bool *, int *, string *, string *, string *, string *);
int SA(double[], bool[], project*, project*, project*, int, double*, double*, double*, double*, double*,
bool, bool, ofstream*, ofstream*, int, double, double[][25], bool);
void tenBinBore(ifstream *, ofstream *, double, double, bool, double, int*, int*, int[][25], int[][25],
int[][15], int[][15], int, project);
void score(int[][25], int[][25], double[][25], int[][15], int[][15], double[][15], int, int);
void BackSelSim(int, project, project, bool[], double[][25], double[][15], ofstream *, randomNum, bool,
bool, string, bool);
void EBackSelSim(int, project, project, bool[], bool, double[][25], double[][15], ofstream *, randomNum,
bool, bool, string, bool);
string itos(int);
string dtos(double);
double roundX(double);

//////////////////////////////////////////////////////////////////////////////
//Model normalization globals. They are modified according to project files.
//////////////////////////////////////////////////////////////////////////////
double cocomoMax = 0.0;
double cocomoMin = 100000000000000000000.0;
double coqualmoMax = 0.0;
double coqualmoMin = 100000000000000000000.0;
double threatMax = 0.0;
double threatMin = 100000000000000000000.0;
double tdevMax = 0.0;
double tdevMin= 100000000000000000000.0;
//////////////////////////////////////////////////////////////////////////////
//Utility weights variables
//////////////////////////////////////////////////////////////////////////////
double alpha = 1.0;
double beta = 1.0;
```

```
double gama = 1.0;
double delta = 1.0;
double relydefect = 1.8; //1.8;
double coolFactor = -10;

////////////////////////////////////////////////////////////////////////////////
//Number of simulations per backselect step
////////////////////////////////////////////////////////////////////////////////
static int simNum=1000;
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
//Arrays to determine search sets for strategic and tactical analysis
////////////////////////////////////////////////////////////////////////////////
bool policySet [25] = {0};
////////////////////////////////////////////////////////////////////////////////

int main(int argc, char* argv[]) {

        double TotalTime, SATime;
        time_t start, end;
        start = clock();

    srand((unsigned)time(0)); //Setting the seed for the random number generator
    cout.setf(ios::fixed,ios::floatfield);
    cout.precision(12);

    ////////////////////////////////////////////////////////////////////////////////
    //Block for declaring variables
    ////////////////////////////////////////////////////////////////////////////////
    project x, s, sb, sn;
    double initattset[x.ATTN];
    bool searchattset[x.ATTN] = {0};
    double finalattset[x.ATTN];
    int attNumUndefined=0;
    double e, eb, en;

    double minE = 100000;
    double maxE = 0;

    minmax stat;
    randomNum rdm;

    double bore = 0.1;
    double emax = 0.0000001; //min energy to be reached before stopping SA
    int kmax = 10000; //max number of itrations for SA
    int run = 0;

    string scoring("");
    string policyVal("");
    string projectVal("");
    string mutation("");
    string stratVal("");
    string bfcVal("");

    bool worst = false;
    bool help = false;
    bool log = false;
    bool file = false;
    bool setPolicy = false;
    bool ERank = false;
    bool OutRuns = false;
    bool Extreme = false;
    bool LCout = false;
    bool LogBest = false;
    bool png = false;
    ////////////////////////////////////////////////////////////////////////////////

    //Parsing command line input
    commandLine(argc, argv, &bore, &emax, &kmax, &worst, &help, &log, &file, &setPolicy, &ERank, &x,
    &OutRuns, &LCout, &Extreme, &LogBest, &png, &run, &policyVal, &projectVal, &stratVal, &bfcVal);

    if (bfcVal.empty()) bfcVal = "bfc";
    if (stratVal.empty()) stratVal = "none";

    if (Extreme) mutation = "extreme";
    else mutation = "normal";

    if (ERank) scoring = "energy";
    else scoring = "bore";

    ////////////////////////////////////////////////////////////////////////////////
    //For command line options that cause termination of STAR
    ////////////////////////////////////////////////////////////////////////////////
    if (help) {
        string helpPath(getenv("HOME"));
        helpPath += "/STAR/help.txt";
```

```
    ifstream helpCheck(helpPath.c_str());

    if (!helpCheck) {
        cout << "Unable to open the help file... exiting..." << endl;
        exit(1);
    }

    string outHelp("cat " + helpPath + "|less");
    system(outHelp.c_str());
    exit(1);
}

//if no input file specified
if (!file) {
    cout << "Please specify a project and the location of its files using -f" << endl
         << "Refer to the help by using -h option" << endl;
    exit(1);
}
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
//Copying original project and setting unset attributes and setting the search set
////////////////////////////////////////////////////////////////////////////////
//Saving the initial settings for the attributes
for (int i=0; i<x.ATTN; i++) {
    initattset[i] = x.attributes[i];
    if (initattset[i] != 0) searchattset[i] = false;
    else searchattset[i] = true;
}

if (setPolicy) {
    for (int i=0; i<25; i++) {
        if (searchattset[i] == true) searchattset[i] = policySet[i];
    }
}

else {
    cout << "Please set the policy to be used";
    exit(1);
}

s=x; //copying the project

//Setting the initial attributes.
for (int i=0; i<25; i++) {
    if (s.attributes[i]==0) {
        s.setattnum(i, rdm.randomGenerateDouble(s.attrangeNorm[0][i], s.attrangeNorm[1][i]));
        attNumUndefined++; //determinning number of undefined attributes
    }
}

for (int i=25; i<x.ATTN; i++) {
    s.setattnum(i, rdm.randomGenerateDouble(s.attrangeNorm[0][i], s.attrangeNorm[1][i]));
}
////////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////
//File I/O Block
////////////////////////////////////////////////////////////////////////////////

time_t rawtime;
    time ( &rawtime );

char * chartime = ctime (&rawtime);
chartime = strtok(chartime, "\n");

char * temptime = strtok(chartime, " :");

string runtime("");

while (temptime!=NULL) {
    runtime = runtime + "_" + temptime;
    temptime = strtok(NULL, " :");
}

    string runspec("");

for (int i=1; i<argc; i++) {
    runspec += argv[i];
    runspec += "_";
}

int f = runspec.find_first_of("STAR");
if (f != 0) runspec = "STAR" + runspec;

runspec = "STAR_results/" + runspec + runtime;
```

```
string htmp(getenv("HOME"));
string hmkdir("");

hmkdir += "mkdir -p " + htmp + "/STAR/" + runspec;
system(hmkdir.c_str());

string hlog(htmp + "/STAR/" + runspec + "/STAR.csv");
string hlogbest(htmp + "/STAR/" + runspec + "/STARbest.csv");

ofstream outfile(hlog.c_str(), ios::out);
outfile.setf(ios::fixed,ios::floatfield);
outfile.precision(12);

if (!outfile) { cout << "Unable to open log file... exiting..." << endl; exit(1); }

for (int i=0; i<x.ATTN; i++) {
    outfile << s.displayAttName(i) << ",";
}
outfile << "Energy\n";

ofstream outbest;
outbest.setf(ios::fixed,ios::floatfield);
outbest.precision(12);

if (LogBest) {
    outbest.close();
    outbest.open(hlogbest.c_str(), ios::out);
    if (!outbest) { cout << "Unable to open log file for best... exiting..." << endl; exit(1); }

    for (int i=0; i<x.ATTN; i++) {
        outbest << s.displayAttName(i) << ",";
    }
    outbest << "Energy\n";
}
else {
    outbest.close();
}
////////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////
//Block for setting and diplaying min/max for the model for the project in
//question. These are used for normalization purposes.
////////////////////////////////////////////////////////////////////////////////
cocomoMax = stat.effortMax(x);
cocomoMin = stat.effortMin(x);
coqualmoMax = stat.defectMax(x);
coqualmoMin =stat.defectMin(x);
threatMax = stat.thrMax(x);
threatMin = stat.thrMin(x);
tdevMax = stat.monthsMax(x);
tdevMin = stat.monthsMin(x);

e = energyM(s);

//Outputting weighting attributes
if (log) {
        cout << "Alpha = " << alpha << endl;
        cout << "Beta  = " << beta << endl;
        cout << "Gamma = " << gama << endl;
        cout << "Delta = " << delta << endl;
        cout << "Rely-Defects = " << relydefect << endl;
        cout << "Cooling Factor = " << coolFactor << endl;


        cout << "Cocomo Min: " << cocomoMin
                << "\nCocomo Max: " << cocomoMax
                << "\nCoqualmo Min: " << coqualmoMin
                << "\nCoqualmo Max: " << coqualmoMax
                << "\nThreat Min: " << threatMin
                << "\nThreat Max: " << threatMax
                << "\nTdev Min : " << tdevMin
                << "\nTdev Max : " << tdevMax;
        ////////////////////////////////////////////////////////////////////////////////

        ////////////////////////////////////////////////////////////////////////////////
        //Finding best or near best solution for project using simulated annealing.
        ////////////////////////////////////////////////////////////////////////////////

        cout << "\nInitial Energy: " << e << endl;
}

sb = s;
eb = e;

double discreteAttEScore [12][25] = {{0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}};
```

```
//Discrete att energy scoring

if (log) cout << "Energy threshold: " << emax << "\nMax iterations: " << kmax << "\nSimulation
iterations: " << simNum << endl;

time_t SAstart, SAend;
SAstart = clock();

int recNum = SA(initattset, searchattset, &s, &sb, &sn, attNumUndefined, &e, &eb, &en, &minE, &maxE,
log, Extreme, &outfile, &outbest, kmax, emax, discreteAttEScore, LogBest);

SAend = clock();
SATime = ((double)(SAend - SAstart)/CLOCKS_PER_SEC );

outfile.close();

for (int i=0; i<x.ATTN; i++)
    finalattset[i] = sb.attributes[i];
////////////////////////////////////////////////////////////////////////////////
//class bore and continuous var 10 bins

int numBest = 0;
int numRest = 0;

//Discrete Scoring
int discreteAttBest [12][25] = {{0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}};
int discreteAttRest [12][25] = {{0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}};
int contAttBest [10][15] = {{0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}};
int contAttRest [10][15] = {{0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}};


ifstream infile(hlog.c_str(), ios::in);
string descLog(htmp + "/STAR/" + runspec + "/STAR-discrete.csv");
outfile.open(descLog.c_str(), ios::out);

if (!ERank) tenBinBore(&infile, &outfile, minE, maxE, worst, bore, &numBest, &numRest,
discreteAttBest, discreteAttRest, contAttBest, contAttRest, recNum, x);

outfile.close();

//end dicretization

//scoring of the attributes
//Discrete Scoring
double discreteAttScore [12][25] = {{0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}};
    double contAttScore [10][15] = {{0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}};

if (!ERank) score(discreteAttBest, discreteAttRest, discreteAttScore, contAttBest, contAttRest,
contAttScore, numBest, numRest);
//end scoring

// Simulations

string simLog(htmp + "/STAR/" + runspec + "/simlog.csv");
outfile.open(simLog.c_str(), ios::out);

if (!ERank) BackSelSim(attNumUndefined, sb, x, searchattset, discreteAttScore, contAttScore, &outfile,
rdm, OutRuns, LCout, runspec, log);
else EBackSelSim(attNumUndefined, sb, x, searchattset, worst, discreteAttEScore, contAttScore,
&outfile, rdm, OutRuns, LCout, runspec, log);
outfile.close();

end = clock();

TotalTime = ((double)(end - start)/CLOCKS_PER_SEC );

string finalout("cd " + htmp + "/STAR/");
string graphPNG("./graphPNG " + runspec);
string graph("./graph " + runspec);
string snum, minEnergy, SANum, TTime, STime, srun, salpha, sbeta, sgamma, sdelta, srelydefect,
scoolFactor,slog;

snum = itos(simNum);
minEnergy = dtos(minE);
SANum = itos(kmax);
TTime = dtos(TotalTime);
STime = dtos(SATime);
srun = itos(run);
salpha = dtos(alpha);
sbeta = dtos(beta);
sgamma = dtos(gama);
sdelta = dtos(delta);
srelydefect = dtos(relydefect);
scoolFactor = dtos(coolFactor);
if (log) slog = "1";
else slog = "0";
```

```
    string policy("gawk -v out=" + runspec + " -v N=" + snum + " -v minEnergy=" + minEnergy + " -v
    scoring=" + scoring + " -v mutation=" + mutation + " -v SANum=" + SANum + " -v TotalTime=" + TTime + "
    -v SATime=" + STime + " -v policyVal=" + policyVal + " -v projectVal=" + projectVal + " -v run=" +
    srun + " -v alpha=" + salpha + " -v beta=" + sbeta + " -v gamma=" + sgamma + " -v delta=" + sdelta + "
    -v relydefect=" + srelydefect + " -v coolFactor=" + scoolFactor + " -v stratVal=" + stratVal + " -v
    bfcVal=" + bfcVal + " -v slog=" + slog + " -f policy.awk " + runspec + "/simlog.csv");

    string command("");

    command += finalout;
    if (png) command += "\n" + graphPNG;
    if (log) command += "\n" + graph;
    command += "\n" + policy;

    int sys = system(command.c_str());

    return 0;
}

double energy(project x) {
    double E = (x.effort()-cocomoMin)/(cocomoMax-cocomoMin);
    double D = (x.defects()-coqualmoMin)/(coqualmoMax-coqualmoMin);
    double T = (threatMax == threatMin) ? 0 : (x.threat()-threatMin)/(threatMax-threatMin);
    double M = (x.months()-tdevMin)/(tdevMax-tdevMin);

    double RD;

    if (relydefect != 0)
        RD = pow(relydefect, (x.getattnum(13)-3));
    else
        RD = 0;

    double EA = E*alpha;
    double DBRD = D*(beta+RD);
    double TG = T*gama;
    double MD = M*delta;

    double BRD = beta+RD;

    double Energy = sqrt(pow(EA, 2)+pow(DBRD, 2)+pow(TG, 2)+pow(MD, 2))/sqrt(pow(alpha, 2)+pow(BRD,
    2)+pow(gama, 2)+pow(delta, 2));
    return Energy;
}

double energyM(project x) {
    double E = (x.effortMutate()-cocomoMin)/(cocomoMax-cocomoMin);
    double D = (x.defectsMutate()-coqualmoMin)/(coqualmoMax-coqualmoMin);
    double T = (threatMax == threatMin) ? 0 : (x.threatMutate()-threatMin)/(threatMax-threatMin);
    double M = (x.monthsMutate()-tdevMin)/(tdevMax-tdevMin);

    double RD;

    if (relydefect != 0)
        RD = pow(relydefect, (x.getattnum(13)-3));
    else
        RD = 0;

    double EA = E*alpha;
    double DBRD = D*(beta+RD);
    double TG = T*gama;
    double MD = M*delta;

    double BRD = beta+RD;

    double EnergyM = sqrt(pow(EA, 2)+pow(DBRD, 2)+pow(TG, 2)+pow(MD, 2))/sqrt(pow(alpha, 2)+pow(BRD,
    2)+pow(gama, 2)+pow(delta, 2));
    return EnergyM;
}

double calcEnergy(project x, double eff, double def, double thr, double mon) {
    double E = (eff-cocomoMin)/(cocomoMax-cocomoMin);
    double D = (def-coqualmoMin)/(coqualmoMax-coqualmoMin);
    double T = (threatMax == threatMin) ? 0 : (thr-threatMin)/(threatMax-threatMin);
    double M = (mon-tdevMin)/(tdevMax-tdevMin);

    double RD;

    if (relydefect != 0)
        RD = pow(relydefect, (x.getattnum(13)-3));
    else
        RD = 0;

    double EA = E*alpha;
    double DBRD = D*(beta+RD);
    double TG = T*gama;
    double MD = M*delta;
```

```
    double BRD = beta+RD;

    double EnergyCalc = sqrt(pow(EA, 2)+pow(DBRD, 2)+pow(TG, 2)+pow(MD, 2))/sqrt(pow(alpha, 2)+pow(BRD,
    2)+pow(gama, 2)+pow(delta, 2));
    return EnergyCalc;
}

double prob(double e, double en, double T) {
    return (double)exp((e-en)/T);
}

project neighbour(project s, double initattset[], int numtochange, bool Extreme) {

    randomNum rdm;
    int numdone=0;
    int ChangedAtt[50] = {0};
    int att;
    double rndRange = rdm.randomN();


    while (numdone<numtochange) {
        bool doneprevious = true;
        while (doneprevious == true) {

            att=rdm.randomGenerateInt(0, 24);

            ChangedAtt[numdone] = att;
            doneprevious = false;
            for (int i=0; i<numdone; i++) {
                if (ChangedAtt[i] == att) {
                    doneprevious = true;
                    break;
                }
            }
        }
        if (initattset[att]==0) {
            if (Extreme) {
                if (rndRange < 0.05) {
                    s.setattnum(att, s.attrangeNorm[0][att]);
                }
                if (rndRange >= 0.05 && rndRange < 0.95) {
                    s.setattnum(att, rdm.randomGenerateDouble(s.attrangeNorm[0][att],
                    s.attrangeNorm[1][att]));
                }
                if (rndRange >= 0.95) {
                    s.setattnum(att, s.attrangeNorm[1][att]);
                }
            }
            else
                s.setattnum(att, rdm.randomGenerateDouble(s.attrangeNorm[0][att],
                s.attrangeNorm[1][att]));
            numdone++;
        }
    }

    for (att=25; att<s.ATTN; att++) {
        s.setattnum(att, rdm.randomGenerateDouble(s.attrangeNorm[0][att], s.attrangeNorm[1][att]));
    }
    return s;
}

double temp(int k, int kmax) {
    return (double)exp(coolFactor*k/kmax);
}

void outStats(project x) {
    cout << "\nCocomoII: " << x.effort() << endl
    << "Coqualmo: " << x.defects() << endl
    << "Threat: " << x.threat() << endl
    << "Tdev: " << x.months() << endl;

}

void partition( double a[], int left, int right, int& lp, int &rp) {
    int i = left + 1;
    int j = left + 1;
    double x = a[left];
    while (j <= right) {
        if (a[j] < x) {
            double temp = a[j];
            a[j] = a[i];
            a[i] = temp;
            i++;
        }
        j++;
    }
    a[left] = a[i-1];
```

```
    a[i-1] = x;
    lp = i - 2;
    rp = i;
}

void quicksort( double a[], int left, int right ) {
    if (left < right) {
        int lp, rp;
        partition( a, left, right, lp, rp );
        quicksort(a, left, lp);
        quicksort(a, rp, right);
    }
}

void sort( double a[], int n ) {
    quicksort(a, 0, n-1);
}

void commandLine(int argc, char* argv[], double* bore, double* emax, int* kmax, bool* worst, bool* help,
        bool* log, bool* file, bool* setPolicy, bool* ERank, project* x, bool* OutRuns, bool* LCout, bool*
        Extreme, bool* LogBest, bool* png, int* run, string* policyVal, string* projectVal, string*
        stratVal, string* bfcVal) {


    for (int i=1; i<argc; i++) {
        string option(argv[i]);

        if (option=="-sim" || option=="-SIM") {
                i++;
            simNum = atoi(argv[i]);
        }

        if (option=="-png" || option=="-PNG") {
                *png = true;
        }

        if (option=="-lb" || option=="-LB") {
                *LogBest = true;
        }

        if (option=="-ex" || option=="-EX")     {
            *Extreme = true;
        }

        if (option=="-al" || option=="-AL")     {
            i++;
            alpha = (double)atof(argv[i]);
        }

        if (option=="-be" || option=="-BE")     {
            i++;
            beta = (double)atof(argv[i]);
        }

        if (option=="-ga" || option=="-GA") {
            i++;
            gama = (double)atof(argv[i]);
        }

        if (option=="-de" || option=="-DE" ) {
            i++;
            delta = (double)atof(argv[i]);
        }

        if (option=="-rd" || option=="-RD" ) {
            i++;
            relydefect = (double)atof(argv[i]);
        }

        if (option=="-cf" || option=="-CF")     {
            i++;
            coolFactor = (double)atof(argv[i]);
        }

        if (option=="-lc" || option=="-LC") {
            *LCout = true;
        }

        if (option=="-or" || option=="-OR") {
            *OutRuns = true;
        }

        if (option=="-n" || option=="-N") {
            *worst = true;
        }

        if (option=="-h" || option=="-H") {
```

```
    *help = true;
}

if (option=="-l" || option=="-L") {
    *log = true;
}

if (option=="-b" || option=="-B") {
    i++;
    *bore = atof(argv[i]);
}

if (option=="-e" || option=="-E") {
    i++;
    *emax = atof(argv[i]);
}

if (option=="-er" || option=="-ER") {
    *ERank = true;
}

if (option=="-run" || option=="-RUN") {
    i++;
    *run = atoi(argv[i]);
}

if (option=="-k" || option=="-K") {
    i++;
    *kmax = atoi(argv[i]);
}

if (option=="-bf" || option=="-BF") {
        alpha = 0.0;
                beta = 2.0;
                gama = 0.0;
                delta = 2.0;
                relydefect = 1.8;
                *bfcVal = option;
                bfcVal->erase(0,1);
}

if (option=="-fc" || option=="-FC") {
        alpha = 2.0;
                beta = 0.0;
                gama = 0.0;
                delta = 2.0;
                relydefect = 0.0;
                *bfcVal = option;
                bfcVal->erase(0,1);
}

if (option=="-bc" || option=="-BC") {
        alpha = 2.0;
                beta = 2.0;
                gama = 0.0;
                delta = 0.0;
                relydefect = 1.8;
                *bfcVal = option;
                bfcVal->erase(0,1);
}

if (option=="-pol" || option=="-POL") {
    *setPolicy = true;
    i++;
    string inPolicy(getenv("HOME"));
    inPolicy += "/STAR/policies/";
    inPolicy += argv[i];
    inPolicy += ".policy";

    *policyVal = argv[i];

    ifstream policy(inPolicy.c_str(), ios::in);
    if (!policy) {
        cout << "Unable to open policy file " << inPolicy << "... exiting..." << endl;
        exit(1);
    }

    string tmpAtt("");

    while (!policy.eof()) {
        policy >> tmpAtt;
        bool right = false;
        for (int f=0;f<25;f++) {
            string ttatt(x->displayAttName(f));
            if (tmpAtt==ttatt) {
                right = true;
                policySet[f] = 1;
```

```
                break;
            }
        }
        if (!right) {
            if (tmpAtt.length() != 0) cout << "The attribute " << tmpAtt << " in " << inPolicy <<
            " is not a valid attribute" << endl;
        }
    }
}

if (option=="-f" || option=="-F") {
    *file = true;
    i++;
    string inProject(getenv("HOME"));
    inProject += "/STAR/STAR_projects/";
    inProject += argv[i];

    *projectVal = argv[i];

    string attvalfn(inProject+".values");
    string attrangefn(inProject+".ranges");

    ifstream attvals(attvalfn.c_str(), ios::in);
    ifstream attranges(attrangefn.c_str(), ios::in);

    if (!attvals) {
        cout << "Unable to open values file for project " << inProject << "... exiting..." << endl;
        exit(1);
    }

    if (!attranges) {
        cout << "Unable to open ranges file for project " << inProject << "... exiting..." << endl;
        exit(1);
    }

    string tmpAtt("");

    while (!attranges.eof()) {
        attranges >> tmpAtt;
        bool right = false;
        for (int f=0;f<x->ATTN;f++) {
            string ttatt(x->displayAttName(f));
            if (tmpAtt==ttatt) {
                right = true;
                attranges >> x->attrangeNorm[0][f] >> x->attrangeNorm[1][f];
            }
        }
        if (!right) {
            double dump;
            attranges >> dump >> dump;
            if (tmpAtt.length() != 0) cout << "The attribute " << tmpAtt << " in " << inProject <<
            ".ranges is not a valid attribute" << endl;
        }
    }

    tmpAtt.clear();

    while (!attvals.eof()) {
        attvals >> tmpAtt;
        bool right = false;
        for (int f=0;f<x->ATTN;f++) {
            string ttatt(x->displayAttName(f));
            if (tmpAtt==ttatt) {
                right = true;
                double tmpval;
                attvals >> tmpval;
                if (tmpval!=0) x->setattnum(f, tmpval);
                x->attrangeNorm[0][f] = x->attrangeNorm[1][f] = tmpval;
            }
        }
        if (!right) {
            double dump;
            attvals >> dump;
            if (tmpAtt.length() != 0) cout << "The attribute " << tmpAtt << " in " << inProject <<
            ".values is not a valid attribute" << endl;
        }
    }
}

if (option=="-st" || option=="-ST") {
    *file = true;
    i++;
    string inStrategy(getenv("HOME"));
    inStrategy += "/STAR/strategies/";
    inStrategy += argv[i];

    *stratVal = argv[i];
```

```
            string attvalfn(inStrategy+".strg");

            ifstream attvals(attvalfn.c_str(), ios::in);

            if (!attvals) {
                cout << "Unable to open definition file for strategy " << inStrategy << "... exiting..."
                << endl;
                exit(1);
            }

            string tmpAtt("");

            while (!attvals.eof()) {
                attvals >> tmpAtt;
                bool right = false;
                for (int f=0;f<x->ATTN;f++) {
                    string ttatt(x->displayAttName(f));
                    if (tmpAtt==ttatt) {
                        right = true;
                        double tmpval;
                        attvals >> tmpval;
                        if (tmpAtt=="Ksloc") {
                                x->attrangeNorm[0][f] = tmpval*x->attrangeNorm[0][f];
                                x->attrangeNorm[1][f] = tmpval*x->attrangeNorm[1][f];
                        }
                        else if (tmpval!=0) {
                                x->setattnum(f, tmpval);
                                x->attrangeNorm[0][f] = x->attrangeNorm[1][f] = tmpval;
                        }
                    }
                }
                if (!right) {
                    double dump;
                    attvals >> dump;
                    if (tmpAtt.length() != 0) cout << "The attribute " << tmpAtt << " in " << inStrategy
                    << ".strg is not a valid attribute" << endl;
                }
            }
        }
    }
}

int SA(double initattset[], bool searchattset[], project *s, project *sb, project *sn, int
        attNumUndefined, double *e, double *eb, double *en, double *minE, double *maxE, bool log, bool
        Extreme, ofstream* outfile, ofstream* outbest, int kmax, double emax, double discreteAttEScore
        [][25], bool LogBest) {
    randomNum rdm;
    int k = 0;
    int bla = 0;
    double discreteAttNum [12][25] = {{0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}};
    *e = energyM(*s);

    *sb = *s;
    *eb = *e;

    int bestNum = 0;

    while (k < kmax && *e > emax) {
        *sn = neighbour(*s, initattset, (int)ceil((double)attNumUndefined/2), Extreme);
        *en = energyM(*sn);

        for (int i=0; i<25; i++) {
            discreteAttEScore[(int)(roundX(sn->attributes[i])*2-1)][i]+=(*en*(1-temp(k, kmax)));
            discreteAttNum[(int)(roundX(sn->attributes[i])*2-1)][i]+=(1-temp(k, kmax));
        }

        if (*en < *eb) {
            *sb = *sn;
            *eb = *en;
            *s = *sn;
            *e = *en;

            if (*en < *minE)
                *minE = *en;
            if (*en > *maxE)
                *maxE = *en;

            bestNum++;
            for (int i=0; i<s->ATTN; i++) {
                *outfile << sn->attributes[i] << ",";
            }
            *outfile << *en << "\n";

            if (LogBest) {
                for (int i=0; i<s->ATTN; i++) {
                    *outbest << sn->attributes[i] << ",";
```

```
                }
                *outbest << *en << "\n";
            }

            k++;
        }
        else if (rdm.randomN() < prob(*e, *en, temp(k, kmax))) {
            bla++;
            *s = *sn;
            *e = *en;

            if (*en < *minE)
                *minE = *en;
            if (*en > *maxE)
                *maxE = *en;

            bestNum++;
            for (int i=0; i<s->ATTN; i++) {
                *outfile << sn->attributes[i] << ",";
            }
            *outfile << *en << "\n";


            k++;
        }
        else
        {

            bestNum++;
            for (int i=0; i<s->ATTN; i++) {
                *outfile << sn->attributes[i] << ",";
            }
            *outfile << *en << "\n";

            k++;
        }

    }

    for (int i=0; i<12; i++) {
        for (int j=0; j<25; j++) {
            if(discreteAttNum[i][j]!=0) discreteAttEScore[i][j]/=discreteAttNum[i][j];
        }
    }

    if (log) {
            cout << "Here is the final output for the unset values:" << endl;
            for (int i=0; i<25; i++) {
                if (searchattset[i]==true) {
                    sb->displayAtt(i);
                }
            }
            for (int i=25; i<s->ATTN; i++) {
                sb->displayAtt(i);
            }
            cout << "Final Energy: " << energy(*sb) << endl;
            cout << "Max Energy: " << *maxE << endl << "Min Energy: " << *minE << endl << "Number of
            Iterations Executed: " << k << endl;
            cout << "Non-Optimal Jumps: " << bla << endl;
    }

    return bestNum;
}

void tenBinBore(ifstream *infile, ofstream *outfile, double minE, double maxE, bool worst, double bore,
        int *numBest, int *numRest, int discreteAttBest[][25], int discreteAttRest[][25], int
         contAttBest[][15], int contAttRest[][15], int recNum, project x) {
    const double zero=0;
    const double one=1;

    string sbuff("");
    char cbuff [50];

    *infile >> sbuff;

    double * energies;
    energies = new double [recNum];
    double borderEnergy;
    double negBorderEnergy;

    int count=0;
    while(!infile->eof()) { //This loop is for getting the energies for evaluation using BORE

        for (int i=0; i<x.ATTN; i++) {
            infile->getline(cbuff, 45, ',');
            sbuff.clear();
            sbuff.append(cbuff);
```

```
        if (sbuff=="") break;
    }

    infile->getline(cbuff, 45);
    sbuff.clear();
    sbuff.append(cbuff);
    if (sbuff=="") break;

    double dbuff = atof(sbuff.c_str());
    dbuff = (dbuff-minE)/(maxE-minE); //Normalising energy

    energies[count] = dbuff;

    count++;
}

sort(energies, recNum);

int BestBorder = (int)floor(recNum*bore);
int NegBestBorder = (int)recNum - (int)ceil(recNum*bore);

borderEnergy = energies[BestBorder];
negBorderEnergy = energies[NegBestBorder];

delete [] energies;
energies = NULL;

infile->clear();
infile->seekg(0, ios::beg);

sbuff.clear();

//Copying the first line to the Discrete output file
*infile >> sbuff;
*outfile << sbuff << "\n";

while(!infile->eof()) { //This is the discretization and scoring loop

    int Att[x.ATTN] = {0};

    for (int i=0; i<25; i++) {
        infile->getline(cbuff, 45, ',');
        sbuff.clear();
        sbuff.append(cbuff);
        if (sbuff=="") break;

        double dbuff = atof(sbuff.c_str());
        dbuff = roundX(dbuff);

        *outfile << dbuff << ",";
        Att[i] = (int)(dbuff*2);
    }

    for (int i=25; i<x.ATTN; i++) {
        infile->getline(cbuff, 45, ',');
        sbuff.clear();
        sbuff.append(cbuff);
        if (sbuff=="") break;

        double dbuff = atof(sbuff.c_str());
        dbuff = (dbuff-x.attrangeNorm[0][i])/(x.attrangeNorm[1][i]-x.attrangeNorm[0][i]);
        dbuff = ceil(dbuff*10);

        *outfile << dbuff << ",";
        Att[i] = (int)dbuff;
    }

    infile->getline(cbuff, 45);
    sbuff.clear();
    sbuff.append(cbuff);
    if (sbuff=="") break;

    double dbuff = atof(sbuff.c_str());
    dbuff = (dbuff-minE)/(maxE-minE);

    if (worst) {
        if (dbuff>negBorderEnergy) {
            *outfile << "1\n";
            *numBest= (*numBest+1);
            for(int i=0; i<25; i++) discreteAttBest [Att[i]-1][i]++;
            for(int i=25; i<x.ATTN; i++) contAttBest [Att[i]-1][i-25]++;
        }
        else {
            *outfile << "0\n";
            *numRest= (*numRest+1);
            for(int i=0; i<25; i++) discreteAttRest [Att[i]-1][i]++;
            for(int i=25; i<x.ATTN; i++) contAttRest [Att[i]-1][i-25]++;
```

```
            }
        }

        else {
            if (dbuff<borderEnergy) {
                *outfile << one << "\n";
                *numBest= (*numBest+1);
                for(int i=0; i<25; i++) discreteAttBest [Att[i]-1][i]++;
                for(int i=25; i<x.ATTN; i++) contAttBest [Att[i]-1][i-25]++;
            }
            else {
                *outfile << zero << "\n";
                *numRest= (*numRest+1);
                for(int i=0; i<25; i++) discreteAttRest [Att[i]-1][i]++;
                for(int i=25; i<x.ATTN; i++) contAttRest [Att[i]-1][i-25]++;
            }
        }
    }
}

void score(int discreteAttBest[][25], int discreteAttRest[][25], double discreteAttScore[][25], int
        contAttBest[][15], int contAttRest[][15], double contAttScore[][15], int numBest, int numRest) {
    for(int i=0; i<25; i++) {
        for (int j=0; j<12; j++) {
            double LBest = ((double)discreteAttBest
            [j][i]/(double)numBest)*((double)numBest/(double)(numBest+numRest));
            double LRest = ((double)discreteAttRest
            [j][i]/(double)numRest)*((double)numRest/(double)(numBest+numRest));
            if ((LBest+LRest)==0) discreteAttScore [j][i] = 0;
            else discreteAttScore [j][i] = pow(LBest, 2)/(LBest+LRest);
        }
    }

   for(int i=0; i<15; i++) {
        for (int j=0; j<10; j++) {
            double LBest = ((double)contAttBest
            [j][i]/(double)numBest)*((double)numBest/(double)(numBest+numRest));
            double LRest = ((double)contAttRest
            [j][i]/(double)numRest)*((double)numRest/(double)(numBest+numRest));
            if ((LBest+LRest)==0) contAttScore [j][i] = 0;
            else contAttScore [j][i] = pow(LBest, 2)/(LBest+LRest);
        }
    }
}

void BackSelSim(int attNumUndefined, project sb, project x, bool searchattset[], double discreteAttScore
        [][25], double contAttScore [][15], ofstream *outfile, randomNum rdm, bool OutRuns, bool LCout,
        string runspec, bool log) {

    int thrownAtt[12][25] = {{0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}};

    if (log) cout << "Attribute,value,score,sumE,sumE^2,medE,spE,sumEffort,sumEffort^2,medEffort,spEffort,
    sumDefects,sumDefects^2,medDefects,spDefects,sumThreat,sumThreat^2,medThreat,spThreat,sumMonths,
    sumMonths^2,medMonths,spMonths\n";

    *outfile << "Attribute,value,score,sumE,sumE^2,medE,spE,sumEffort,sumEffort^2,medEffort,spEffort,
    sumDefects,sumDefects^2,medDefects,spDefects,sumThreat,sumThreat^2,medThreat,spThreat,sumMonths,
    sumMonths^2,medMonths,spMonths\n";

    int numPolicy = 0;

    for (int k=0; k<(attNumUndefined*12); k++) {

        double minScore = 100000;
        int minAtt;
        int minAttVal;

        for (int i=0; i<25; i++) {

            for (int j=(int)(roundX(x.attrangeNorm[0][i])*2); j<=(int)(roundX(x.attrangeNorm[1][i])*2);
            j++) {

                    if (searchattset[i] == true && thrownAtt[j-1][i] == 0 && discreteAttScore [j-1][i]
                    < minScore && discreteAttScore [j-1][i] != 0) {
                minScore = discreteAttScore [j-1][i];
                minAtt=i;
                minAttVal=j-1;
                }
            }
        }

        thrownAtt[minAttVal][minAtt] ++;

        if (discreteAttScore [minAttVal][minAtt] != 0 && thrownAtt[minAttVal][minAtt] == 1) {

            numPolicy++;
            ofstream timfile;
```

```
timfile.setf(ios::fixed,ios::floatfield);
timfile.precision(12);

if (LCout) {
    //Tim Out
    ////////////////////////////////////////////////////
    char c[10];
    sprintf(c, "%i", numPolicy);
    string numc(c);

    string htmp(getenv("HOME"));
    string timLog(htmp + "/STAR/" + runspec + "/LCcomp" + numc);
    timfile.open(timLog.c_str(), ios::app);
    ////////////////////////////////////////////////////
}

double avgE = 0;
double sdE = 0;
double medianE = 0;
double spreadE = 0;

double * E;
E = new double [simNum];

double avgEffort = 0;
double sdEffort = 0;
double medianEffort = 0;
double spreadEffort = 0;

double * Effort;
Effort = new double [simNum];

double avgDefects = 0;
double sdDefects = 0;
double medianDefects = 0;
double spreadDefects = 0;

double * Defects;
Defects = new double [simNum];

double avgThreat = 0;
double sdThreat = 0;
double medianThreat = 0;
double spreadThreat = 0;

double * Threat;
Threat = new double [simNum];

double avgMonths = 0;
double sdMonths = 0;
double medianMonths = 0;
double spreadMonths = 0;

double * Months;
Months = new double [simNum];

for (int simn=0; simn<simNum; simn++) {

    if (LCout) {
        //Tim out
        ////////////////////////////////////////////////////
        if (simn == 0) {
            timfile << x.displayAttName(3);
            for (int i=4; i<25; i++)
                timfile << "," << x.displayAttName(i);
            for (int i=37; i<x.ATTN; i++)
                timfile << "," << x.displayAttName(i);
            timfile << ",Effort\n";
        }
        ////////////////////////////////////////////////////
    }

    project sim = sb; //Starting from the best SA case
    for (int i=0; i<25; i++) {
        bool allThrown = true;
        int numNotThrown = 0;
        double notThrown [12] = {0};

        for (int j=(int)(roundX(x.attrangeNorm[0][i])*2);
        j<=(int)(roundX(x.attrangeNorm[1][i])*2); j++) {
            if(thrownAtt[j-1][i]==0) {
                    allThrown = false;
                    notThrown[numNotThrown]=(double)j/2;
                    numNotThrown++;
            }
        }
```

```
                if(allThrown) sim.setattnum(i, roundX(rdm.randomGenerateDouble(x.attrangeNorm[0][i],
                x.attrangeNorm[1][i])));

                else sim.setattnum(i, notThrown[rdm.randomGenerateInt(0,numNotThrown-1)]);
            }

            for (int l=25; l<x.ATTN; l++) {
                sim.setattnum(l, rdm.randomGenerateDouble(x.attrangeNorm[0][l], x.attrangeNorm[1][l]));
            }

            double tmpEffort = sim.effortMutate();
            double tmpDefects = sim.defectsMutate();
            double tmpThreat = sim.threatMutate();
            double tmpMonths = sim.monthsMutate();
            double tmpE = calcEnergy(sim, tmpEffort, tmpDefects, tmpThreat, tmpMonths);

            if (LCout) {
                //Tim Out
                ////////////////////////////////////////////////////////////
                timfile << sim.getattnumvaleff(1);//getattnum(1);
                for (int i=2; i<23; i++)
                    timfile << "," << sim.getattnumvaleff(i);//getattnum(i);
                for (int i=37; i<x.ATTN; i++)
                    timfile << "," << sim.getattnum(i);
                timfile << "," << sim.effort() << "\n";
                ////////////////////////////////////////////////////////////
            }

            avgE += tmpE;
            sdE += tmpE*tmpE;
            E[simn] = tmpE;

            avgEffort += tmpEffort;
            sdEffort += tmpEffort*tmpEffort;
            Effort[simn] = tmpEffort;

            avgDefects += tmpDefects;
            sdDefects += tmpDefects*tmpDefects;
            Defects[simn] = tmpDefects;

            avgThreat += tmpThreat;
            sdThreat += tmpThreat*tmpThreat;
            Threat[simn] = tmpThreat;

            avgMonths += tmpMonths;
            sdMonths += tmpMonths*tmpMonths;
            Months[simn] = tmpMonths;
        }

        sort(E, simNum);
        sort(Effort, simNum);
        sort(Defects, simNum);
        sort(Threat, simNum);
        sort(Months, simNum);

        if (OutRuns) {
            char c[10];
            sprintf(c, "%i", k);
            string numc(c);

            string htmp(getenv("HOME"));
            string runLog(htmp + "/STAR/" + runspec + "/run" + numc);
            ofstream runfile(runLog.c_str(), ios::out);
            runfile.setf(ios::fixed,ios::floatfield);
            runfile.precision(12);

            for (int i=0; i<simNum; i++) {
                runfile << E[i] << "\t" << Effort[i] << "\t" << Defects[i] << "\t" << Threat[i] <<
                "\t" << Months[i] << "\n";
            }
        }

//          avgE = avgE/simNum;
//          sdE = sqrt(fabs((sdE/simNum)-(avgE*avgE)));
        medianE = E[(int)(simNum/2)];
        spreadE = E[(int)(simNum*3/4)]-E[(int)(simNum/2)];

//          avgEffort = avgEffort/simNum;
//          sdEffort = sqrt(fabs((sdEffort/simNum)-(avgEffort*avgEffort)));
        medianEffort = Effort[(int)(simNum/2)];
        spreadEffort = Effort[(int)(simNum*3/4)]-Effort[(int)(simNum/2)];

//          avgDefects = avgDefects/simNum;
//          sdDefects = sqrt(fabs((sdDefects/simNum)-(avgDefects*avgDefects)));
        medianDefects = Defects[(int)(simNum/2)];
        spreadDefects = Defects[(int)(simNum*3/4)]-Defects[(int)(simNum/2)];
```

```
//              avgThreat = avgThreat/simNum;
//              sdThreat = sqrt(fabs((sdThreat/simNum)-(avgThreat*avgThreat)));
            medianThreat = Threat[(int)(simNum/2)];
            spreadThreat = Threat[(int)(simNum*3/4)]-Threat[(int)(simNum/2)];

//              avgMonths = avgMonths/simNum;
//              sdMonths = sqrt(fabs((sdMonths/simNum)-(avgMonths*avgMonths)));
            medianMonths = Months[(int)(simNum/2)];
            spreadMonths = Months[(int)(simNum*3/4)]-Months[(int)(simNum/2)];

            //Output of simulation results
            if (log) {
                    cout << x.displayAttName(minAtt) << ","
                    << (double)(minAttVal+1)/2 << "," << discreteAttScore [minAttVal][minAtt]   << ","
                    << avgE << "," << sdE << "," << medianE << "," << spreadE << ","
                    << avgEffort << "," << sdEffort << "," << medianEffort << "," << spreadEffort << ","
                    << avgDefects << "," << sdDefects << "," << medianDefects << "," << spreadDefects <<
                     "," << avgThreat << "," << sdThreat << "," << medianThreat << "," << spreadThreat <<
                     "," << avgMonths << "," << sdMonths << "," << medianMonths << "," << spreadMonths <<
                     "\n";
            }
            *outfile << x.displayAttName(minAtt) << ","
            << (double)(minAttVal+1)/2 << "," << discreteAttScore [minAttVal][minAtt]   << ","
            << avgE << "," << sdE << "," << medianE << "," << spreadE << ","
            << avgEffort << "," << sdEffort << "," << medianEffort << "," << spreadEffort << ","
            << avgDefects << "," << sdDefects << "," << medianDefects << "," << spreadDefects << ","
            << avgThreat << "," << sdThreat << "," << medianThreat << "," << spreadThreat << ","
            << avgMonths << "," << sdMonths << "," << medianMonths << "," << spreadMonths << "\n";

            delete [] E;
            E = NULL;

            delete [] Effort;
            Effort = NULL;

            delete [] Defects;
            Defects = NULL;

            delete [] Threat;
            Threat = NULL;

            delete [] Months;
            Months = NULL;
        }
    }
}

void EBackSelSim(int attNumUndefined, project sb, project x, bool searchattset[], bool worst, double
        discreteAttEScore [][25], double contAttScore [][15], ofstream *outfile, randomNum rdm, bool
        OutRuns, bool LCout, string runspec, bool log) {

    int thrownAtt[12][25] = {{0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}};

    if (log) {
            cout << "Energy Ranking:" << endl;
            cout << "Attribute,value,score,sumE,sumE^2,medE,spE,sumEffort,sumEffort^2,medEffort,spEffort,
            sumDefects,sumDefects^2,medDefects,spDefects,sumThreat,sumThreat^2,medThreat,spThreat,
            sumMonths,sumMonths^2,medMonths,spMonths\n";
    }

    *outfile << "Attribute,value,score,sumE,sumE^2,medE,spE,sumEffort,sumEffort^2,medEffort,spEffort,
    sumDefects,sumDefects^2,medDefects,spDefects,sumThreat,sumThreat^2,medThreat,spThreat,sumMonths,
    sumMonths^2,medMonths,spMonths\n";

    int numPolicy = 0;

    for (int k=0; k<(attNumUndefined*12); k++) {

        double minScore = -1;
        int minAtt;
        int minAttVal;

        if (worst) {
            minScore = 100000000;
        }

        for (int i=0; i<25; i++) {
            for (int j=(int)roundX(x.attrangeNorm[0][i])*2; j<=(int)roundX(x.attrangeNorm[1][i])*2; j++) {
                if (!worst && searchattset[i] == true && thrownAtt[j-1][i] == 0 && discreteAttEScore
                [j-1][i] > minScore && discreteAttEScore [j-1][i] != 0) {
                    minScore = discreteAttEScore [j-1][i];
                    minAtt=i;
                    minAttVal=j-1;
                }
                else if (worst && searchattset[i] == true && thrownAtt[j-1][i] == 0 && discreteAttEScore
                [j-1][i] < minScore && discreteAttEScore [j-1][i] != 0) {
                    minScore = discreteAttEScore [j-1][i];
```

```
            minAtt=i;
            minAttVal=j-1;
        }
    }
}

thrownAtt[minAttVal][minAtt] ++;

if (discreteAttEScore [minAttVal][minAtt] != 0 && thrownAtt[minAttVal][minAtt] == 1) {

    numPolicy++;
    ofstream timfile;
    timfile.setf(ios::fixed,ios::floatfield);
    timfile.precision(12);

    if (LCout) {
        //Tim Out
        ////////////////////////////////////////////////////
        char c[10];
        sprintf(c, "%i", numPolicy);
        string numc(c);

        string htmp(getenv("HOME"));
        string timLog(htmp + "/STAR/" + runspec + "/LCcomp" + numc);
        timfile.open(timLog.c_str(), ios::app);
        ////////////////////////////////////////////////////
    }

    double avgE = 0;
    double sdE = 0;
    double medianE = 0;
    double spreadE = 0;

    double * E;
    E = new double [simNum];

    double avgEffort = 0;
    double sdEffort = 0;
    double medianEffort = 0;
    double spreadEffort = 0;

    double * Effort;
    Effort = new double [simNum];

    double avgDefects = 0;
    double sdDefects = 0;
    double medianDefects = 0;
    double spreadDefects = 0;

    double * Defects;
    Defects = new double [simNum];

    double avgThreat = 0;
    double sdThreat = 0;
    double medianThreat = 0;
    double spreadThreat = 0;

    double * Threat;
    Threat = new double [simNum];

    double avgMonths = 0;
    double sdMonths = 0;
    double medianMonths = 0;
    double spreadMonths = 0;

    double * Months;
    Months = new double [simNum];

    for (int simn=0; simn<simNum; simn++) {

        if (LCout) {
            //Tim out
            ////////////////////////////////////////////////////
            if (simn == 0) {
                timfile << x.displayAttName(3);
                for (int i=4; i<25; i++)
                    timfile << "," << x.displayAttName(i);
                for (int i=37; i<x.ATTN; i++)
                    timfile << "," << x.displayAttName(i);
                timfile << ",Effort\n";
            }
            ////////////////////////////////////////////////////
        }

        project sim = sb; //Starting from the best SA case
        for (int i=0; i<25; i++) {
            bool allThrown = true;
```

```
            int numNotThrown = 0;
            double notThrown [12] = {0};

            for (int j=(int)(roundX(x.attrangeNorm[0][i])*2);
            j<=(int)(roundX(x.attrangeNorm[1][i])*2); j++) {
                if(thrownAtt[j-1][i]==0) {
                        allThrown = false;
                        notThrown[numNotThrown]=(double)j/2;
                        numNotThrown++;
                }
            }

            if(allThrown) sim.setattnum(i, roundX(rdm.randomGenerateDouble(x.attrangeNorm[0][i],
            x.attrangeNorm[1][i])));

            else sim.setattnum(i, notThrown[rdm.randomGenerateInt(0,numNotThrown-1)]);
        }

        for (int l=25; l<x.ATTN; l++) {
            sim.setattnum(l, rdm.randomGenerateDouble(x.attrangeNorm[0][l], x.attrangeNorm[1][l]));
        }

        double tmpEffort = sim.effortMutate();
        double tmpDefects = sim.defectsMutate();
        double tmpThreat = sim.threatMutate();
        double tmpMonths = sim.monthsMutate();

        double tmpE = calcEnergy(sim, tmpEffort, tmpDefects, tmpThreat, tmpMonths);

        if (LCout) {
            //Tim Out
            ////////////////////////////////////////////////////////
            timfile << sim.getattnumvaleff(1);//getattnum(1);
            for (int i=2; i<23; i++)
                timfile << "," << sim.getattnumvaleff(i);//getattnum(i);
            for (int i=37; i<x.ATTN; i++)
                timfile << "," << sim.getattnum(i);
            timfile << "," << sim.effort() << "\n";
            ////////////////////////////////////////////////////////
        }

        avgE += tmpE;
        sdE += tmpE*tmpE;
        E[simn] = tmpE;

        avgEffort += tmpEffort;
        sdEffort += tmpEffort*tmpEffort;
        Effort[simn] = tmpEffort;

        avgDefects += tmpDefects;
        sdDefects += tmpDefects*tmpDefects;
        Defects[simn] = tmpDefects;

        avgThreat += tmpThreat;
        sdThreat += tmpThreat*tmpThreat;
        Threat[simn] = tmpThreat;

        avgMonths += tmpMonths;
        sdMonths += tmpMonths*tmpMonths;
        Months[simn] = tmpMonths;
    }

    sort(E, simNum);
    sort(Effort, simNum);
    sort(Defects, simNum);
    sort(Threat, simNum);
    sort(Months, simNum);

    if (OutRuns) {
        char c[10];
        sprintf(c, "%i", k);
        string numc(c);

        string htmp(getenv("HOME"));
        string runLog(htmp + "/STAR/" + runspec + "/run" + numc);
        ofstream runfile(runLog.c_str(), ios::out);
        runfile.setf(ios::fixed,ios::floatfield);
        runfile.precision(12);

        for (int i=0; i<simNum; i++) {
            runfile << E[i] << "\t" << Effort[i] << "\t" << Defects[i] << "\t" << Threat[i] <<
            "\t" << Months[i] << "\n";
        }
    }

    medianE = E[(int)(simNum/2)];
    spreadE = E[(int)(simNum*3/4)]-E[(int)(simNum/2)];
```

```
            medianEffort = Effort[(int)(simNum/2)];
            spreadEffort = Effort[(int)(simNum*3/4)]-Effort[(int)(simNum/2)];

            medianDefects = Defects[(int)(simNum/2)];
            spreadDefects = Defects[(int)(simNum*3/4)]-Defects[(int)(simNum/2)];

            medianThreat = Threat[(int)(simNum/2)];
            spreadThreat = Threat[(int)(simNum*3/4)]-Threat[(int)(simNum/2)];

            medianMonths = Months[(int)(simNum/2)];
            spreadMonths = Months[(int)(simNum*3/4)]-Months[(int)(simNum/2)];

            //Output of simulation results
            if (log) {
                    cout << x.displayAttName(minAtt) << ","
                    << (double)(minAttVal+1)/2 << "," << discreteAttEScore [minAttVal][minAtt] << ","
                    << avgE << "," << sdE << "," << medianE << "," << spreadE << ","
                    << avgEffort << "," << sdEffort << "," << medianEffort << "," << spreadEffort << ","
                    << avgDefects << "," << sdDefects << "," << medianDefects << "," << spreadDefects <<
                    "," << avgThreat << "," << sdThreat << "," << medianThreat << "," << spreadThreat <<
                    "," << avgMonths << "," << sdMonths << "," << medianMonths << "," << spreadMonths
                    <<"\n";
            }

            *outfile << x.displayAttName(minAtt) << ","
            << (double)(minAttVal+1)/2 << "," << discreteAttEScore [minAttVal][minAtt] << ","
            << avgE << "," << sdE << "," << medianE << "," << spreadE << ","
            << avgEffort << "," << sdEffort << "," << medianEffort << "," << spreadEffort << ","
            << avgDefects << "," << sdDefects << "," << medianDefects << "," << spreadDefects << ","
            << avgThreat << "," << sdThreat << "," << medianThreat << "," << spreadThreat << ","
            << avgMonths << "," << sdMonths << "," << medianMonths << "," << spreadMonths <<"\n";

            delete [] E;
            E = NULL;

            delete [] Effort;
            Effort = NULL;

            delete [] Defects;
            Defects = NULL;

            delete [] Threat;
            Threat = NULL;

            delete [] Months;
            Months = NULL;
        }
    }
}

string itos(int i)      // convert int to string
{
        stringstream s;
        s << i;
        return s.str();
}

string dtos(double d)   // convert double to string
{
        stringstream s;
        s.setf(ios::fixed,ios::floatfield);
        s.precision(12);
        s << d;
        return s.str();
}

double roundX(double num)
{
        double decimal = num - (int)num;

        if (decimal < 0.25) num = (int)num;
        else if (decimal >= 0.25 && decimal < 0.75) num = (int)num + 0.5;
        else num = (int)num + 1;

        return num;
}
```

B.2 project.cpp

```
#include "project.h"

project::project() {
    initialize();
```

```
}

project::~project() {
}
void project::initialize() {
    //initialization

    ////////////////////////////////////////////////////////////////////////////////
//Attribute and slope ranges. This is modified according to project files
////////////////////////////////////////////////////////////////////////////////
        double attrangeNormTemp[] = {1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 0.000, -0.183, 0.08, 0.00, -0.208, 0.100, 0.00, -0.190, 0.110, -1.560, 0.073, -0.178, 2.00,
          0.55, 1, 3, 0.28, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 5, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6,
           0.112, -0.035, 0.14, 0.14, -0.048, 0.156, 0.14, -0.053, 0.176, -1.014, 0.210, -0.078, 11.0,
            0.997973, 980, 3.67, 0.33};

        for (int i=0; i<2; i++) {
                for (int j=0; j<ATTN; j++) {
                        attrangeNorm[i][j]= attrangeNormTemp[ATTN*i+j];
                }
        }

    pRSlope=0.052;
    nRSlope=-0.100;
    dfrRSlope=0.113;

    pDSlope=0.072;
    nDSlope=-0.105;
    dfrDSlope=0.132;

    pCSlope=0.072;
    nCSlope=-0.106;
    dfrCSlope=0.151;

    SFSlope=-1.265;
    posEMSlope=0.140;
    negEMSlope=-0.109;

    A=6.51;
    B=0.77;
    C=3.335;
    D=0.305;

    ksloc=100;

    for (int i=0; i<25; i++)
        attributes[i]=0;


    //defect removal schemes
    automated_analysis=0;
    peer_reviews=0;
    execution_testing_and_tools=0;

    //scale factors
    prec=0;
    flex=0;
    resl=0;
    team=0;
    pmat=0;

    //effort multipliers
    time=0;
    stor=0;
    data=0;
    pvol=0;
    ruse=0;
    rely=0;
    docu=0;
    acap=0;
    pcap=0;
    pcon=0;
    apex=0;
    plex=0;
    ltex=0;
    tool=0;
    sced=0;
    cplx=0;
    site=0;

    pCoqualRSlope=pRSlope;
    nCoqualRSlope=nRSlope;
    dfrCoqualRSlope=dfrRSlope;

    pCoqualDSlope=pDSlope;
    nCoqualDSlope=nDSlope;
    dfrCoqualDSlope=dfrDSlope;
```

```
    pCoqualCSlope=pCSlope;
    nCoqualCSlope=nCSlope;
    dfrCoqualCSlope=dfrCSlope;

    SFCocomoSlope=SFSlope;
    posEMCocomoSlope=posEMSlope;
    negEMCocomoSlope=negEMSlope;
}
//The regulated CocomoII model
double project::SFCocomo(double rating) {return (double)(rating-6)*SFCocomoSlope;}
double project::posEMCocomo(double rating) {return (double)(rating-3)*posEMCocomoSlope + 1;}
double project::negEMCocomo(double rating) {return (double)(rating-3)*negEMCocomoSlope + 1;}

//cocomoII stuff
double project::fprec() {return SFCocomo(prec);}
double project::fflex() {return SFCocomo(flex);}
double project::fresl() {return SFCocomo(resl);}
double project::fteam() {return SFCocomo(team);}
double project::fpmat() {return SFCocomo(pmat);}
double project::ftime() {return posEMCocomo(time);}
double project::fstor() {return posEMCocomo(stor);}
double project::fdata() {return posEMCocomo(data);}
double project::fpvol() {return posEMCocomo(pvol);}
double project::fruse() {return posEMCocomo(ruse);}
double project::frely() {
        int relyLB = rely + (automated_analysis-3>0? automated_analysis-3 : 0)
        + (execution_testing_and_tools-3>0? execution_testing_and_tools-3 : 0)
        + (peer_reviews-3>0? peer_reviews-3 : 0);
        return posEMCocomo(5>relyLB? relyLB : 5);}
double project::fcplx() {return posEMCocomo(cplx);}
double project::fdocu() {return posEMCocomo(docu);}
double project::fsced() {return negEMCocomo(sced);}
double project::facap() {return negEMCocomo(acap);}
double project::fpcap() {return negEMCocomo(pcap);}
double project::fpcon() {return negEMCocomo(pcon);}
double project::fapex() {return negEMCocomo(apex);}
double project::fltex() {return negEMCocomo(ltex);}
double project::ftool() {
        int toolLB = tool + (automated_analysis-3>0? automated_analysis-3 : 0)
                + (execution_testing_and_tools-3>0? execution_testing_and_tools-3 : 0);
        return negEMCocomo(5>toolLB? toolLB : 5);}
double project::fsite() {return negEMCocomo(site);}
double project::fplex() {return negEMCocomo(plex);}

double project::getattnum(int attnum) {
    switch (attnum) {
        case 0: return automated_analysis; break;
        case 1: return peer_reviews; break;
        case 2: return execution_testing_and_tools; break;
        case 3: return prec; break;
        case 4: return flex; break;
        case 5: return resl; break;
        case 6: return team; break;
        case 7: return pmat; break;
        case 8: return time; break;
        case 9: return stor; break;
        case 10: return data; break;
        case 11: return pvol; break;
        case 12: return ruse; break;
        case 13: return rely; break;
        case 14: return docu; break;
        case 15: return acap; break;
        case 16: return pcap; break;
        case 17: return pcon; break;
        case 18: return apex; break;
        case 19: return plex; break;
        case 20: return ltex; break;
        case 21: return tool; break;
        case 22: return sced; break;
        case 23: return cplx; break;
        case 24: return site; break;
        //Slopes, A, B and Ksloc
        case 25: return pCoqualRSlope; break;
        case 26: return nCoqualRSlope; break;
        case 27: return dfrCoqualRSlope; break;
        case 28: return pCoqualDSlope; break;
        case 29: return nCoqualDSlope; break;
        case 30: return dfrCoqualDSlope; break;
        case 31: return pCoqualCSlope; break;
        case 32: return nCoqualCSlope; break;
        case 33: return dfrCoqualCSlope; break;
        case 34: return SFCocomoSlope; break;
        case 35: return posEMCocomoSlope; break;
        case 36: return negEMCocomoSlope; break;
        case 37: return A; break;
        case 38: return B; break;
```

```
        case 39: return ksloc; break;
        case 40: return C; break;
        case 41: return D; break;
        default: return 0.0; break;
    }
}

double project::getattnumvaleff(int attnum) {
    switch (attnum) {
        case 1: return fprec(); break;
        case 2: return fflex(); break;
        case 3: return fresl(); break;
        case 4: return fteam(); break;
        case 5: return fpmat(); break;
        case 6: return ftime(); break;
        case 7: return fstor(); break;
        case 8: return fdata(); break;
        case 9: return fpvol(); break;
        case 10: return fruse(); break;
        case 11: return frely(); break;
        case 12: return fdocu(); break;
        case 13: return facap(); break;
        case 14: return fpcap(); break;
        case 15: return fpcon(); break;
        case 16: return fapex(); break;
        case 17: return fplex(); break;
        case 18: return fltex(); break;
        case 19: return ftool(); break;
        case 20: return fsced(); break;
        case 21: return fcplx(); break;
        case 22: return fsite(); break;
        default: return 0.0; break;
    }
}

double project::getattnumvaldefR(int attnum) {
    switch (attnum) {
        case 0: return automated_analysisR(); break;
        case 1: return peer_reviewsR(); break;
        case 2: return execution_testing_and_toolsR(); break;
        case 3: return precR(); break;
        case 4: return flexR(); break;
        case 5: return reslR(); break;
        case 6: return teamR(); break;
        case 7: return pmatR(); break;
        case 8: return timeR(); break;
        case 9: return storR(); break;
        case 10: return dataR(); break;
        case 11: return pvolR(); break;
        case 12: return ruseR(); break;
        case 13: return relyR(); break;
        case 14: return docuR(); break;
        case 15: return acapR(); break;
        case 16: return pcapR(); break;
        case 17: return pconR(); break;
        case 18: return apexR(); break;
        case 19: return plexR(); break;
        case 20: return ltexR(); break;
        case 21: return toolR(); break;
        case 22: return scedR(); break;
        case 23: return cplxR(); break;
        case 24: return siteR(); break;
        default: return 0.0; break;
    }
}

double project::getattnumvaldefD(int attnum) {
    switch (attnum) {
        case 0: return automated_analysisD(); break;
        case 1: return peer_reviewsD(); break;
        case 2: return execution_testing_and_toolsD(); break;
        case 3: return precD(); break;
        case 4: return flexD(); break;
        case 5: return reslD(); break;
        case 6: return teamD(); break;
        case 7: return pmatD(); break;
        case 8: return timeD(); break;
        case 9: return storD(); break;
        case 10: return dataD(); break;
        case 11: return pvolD(); break;
        case 12: return ruseD(); break;
        case 13: return relyD(); break;
        case 14: return docuD(); break;
        case 15: return acapD(); break;
        case 16: return pcapD(); break;
        case 17: return pconD(); break;
        case 18: return apexD(); break;
```

```
        case 19: return plexD(); break;
        case 20: return ltexD(); break;
        case 21: return toolD(); break;
        case 22: return scedD(); break;
        case 23: return cplxD(); break;
        case 24: return siteD(); break;
        default: return 0.0; break;
    }
}

double project::getattnumvaldefC(int attnum) {
    switch (attnum) {
        case 0: return automated_analysisC(); break;
        case 1: return peer_reviewsC(); break;
        case 2: return execution_testing_and_toolsC(); break;
        case 3: return precC(); break;
        case 4: return flexC(); break;
        case 5: return reslC(); break;
        case 6: return teamC(); break;
        case 7: return pmatC(); break;
        case 8: return timeC(); break;
        case 9: return storC(); break;
        case 10: return dataC(); break;
        case 11: return pvolC(); break;
        case 12: return ruseC(); break;
        case 13: return relyC(); break;
        case 14: return docuC(); break;
        case 15: return acapC(); break;
        case 16: return pcapC(); break;
        case 17: return pconC(); break;
        case 18: return apexC(); break;
        case 19: return plexC(); break;
        case 20: return ltexC(); break;
        case 21: return toolC(); break;
        case 22: return scedC(); break;
        case 23: return cplxC(); break;
        case 24: return siteC(); break;
        default: return 0.0; break;
    }
}

void project::setattnum(int attnum, double rating) {
    switch (attnum) {
        case 0: sautomated_analysis(rating); break;
        case 1: speer_reviews(rating); break;
        case 2: sexecution_testing_and_tools(rating); break;
        case 3: sprec(rating); break;
        case 4: sflex(rating); break;
        case 5: sresl(rating); break;
        case 6: steam(rating); break;
        case 7: spmat(rating); break;
        case 8: stime(rating); break;
        case 9: sstor(rating); break;
        case 10: sdata(rating); break;
        case 11: spvol(rating); break;
        case 12: sruse(rating); break;
        case 13: srely(rating); break;
        case 14: sdocu(rating); break;
        case 15: sacap(rating); break;
        case 16: spcap(rating); break;
        case 17: spcon(rating); break;
        case 18: sapex(rating); break;
        case 19: splex(rating); break;
        case 20: sltex(rating); break;
        case 21: stool(rating); break;
        case 22: ssced(rating); break;
        case 23: scplx(rating); break;
        case 24: ssite(rating); break;
        //Slopes, A, B and Ksloc
        case 25: spCoqualRSlope(rating); break;
        case 26: snCoqualRSlope(rating);break;
        case 27: sdfrCoqualRSlope(rating);break;
        case 28: spCoqualDSlope(rating);break;
        case 29: snCoqualDSlope(rating);break;
        case 30: sdfrCoqualDSlope(rating);break;
        case 31: spCoqualCSlope(rating);break;
        case 32: snCoqualCSlope(rating);break;
        case 33: sdfrCoqualCSlope(rating);break;
        case 34: sSFCocomoSlope(rating);break;
        case 35: sposEMCocomoSlope(rating);break;
        case 36: snegEMCocomoSlope(rating);break;
        case 37: sA(rating); break;
        case 38: sB(rating); break;
        case 39: sKsloc(rating); break;
        case 40: sC(rating); break;
        case 41: sD(rating); break;
    }
```

```
}

double project::setatt(double rating, double min, double max) {
    if (rating > max) return max;
    else if (rating < min) return min;
    else return rating;
}

void project::sprec(double rating) {
    double min=1;double max=6;
    attributes[3]=prec=setatt(rating, min, max);
}
void project::sflex(double rating) {
    double min=1;double max=6;
    attributes[4]=flex=setatt(rating, min, max);
}
void project::sresl(double rating) {
    double min=1;double max=6;
    attributes[5]=resl=setatt(rating, min, max);
}
void project::steam(double rating) {
    double min=1;double max=6;
    attributes[6]=team=setatt(rating, min, max);
}
void project::spmat(double rating) {
    double min=1;double max=6;
    attributes[7]=pmat=setatt(rating, min, max);
}

void project::stime(double rating) {
    double min=3;double max=6;
    attributes[8]=time=setatt(rating, min, max);
}
void project::sstor(double rating) {
    double min=3;double max=6;
    attributes[9]=stor=setatt(rating, min, max);
}
void project::sdata(double rating) {
    double min=2;double max=5;
    attributes[10]=data=setatt(rating, min, max);
}
void project::spvol(double rating) {
    double min=2;double max=5;
    attributes[11]=pvol=setatt(rating, min, max);
}
void project::sruse(double rating) {
    double min=2;double max=6;
    attributes[12]=ruse=setatt(rating, min, max);
}
void project::srely(double rating) {
    double min=1;double max=5;
    attributes[13]=rely=setatt(rating, min, max);
}
void project::ssced(double rating) {
    double min=1;double max=5;
    attributes[22]=sced=setatt(rating, min, max);
}
void project::scplx(double rating) {
    double min=1;double max=6;
    attributes[23]=cplx=setatt(rating, min, max);
}
void project::sdocu(double rating) {
    double min=1;double max=5;
    attributes[14]=docu=setatt(rating, min, max);
}

void project::sacap(double rating) {
    double min=1;double max=5;
    attributes[15]=acap=setatt(rating, min, max);
}
void project::spcap(double rating) {
    double min=1;double max=5;
    attributes[16]=pcap=setatt(rating, min, max);
}
void project::spcon(double rating) {
    double min=1;double max=5;
    attributes[17]=pcon=setatt(rating, min, max);
}
void project::sapex(double rating) {
    double min=1;double max=5;
    attributes[18]=apex=setatt(rating, min, max);
}
void project::sltex(double rating) {
    double min=1;double max=5;
    attributes[20]=ltex=setatt(rating, min, max);
}
void project::stool(double rating) {
```

```
    double min=1;double max=5;
    attributes[21]=tool=setatt(rating, min, max);
}
void project::ssite(double rating) {
    double min=1;double max=6;
    attributes[24]=site=setatt(rating, min, max);
}
void project::splex(double rating) {
    double min=1;double max=5;
    attributes[19]=plex=setatt(rating, min, max);
}

void project::sautomated_analysis(double rating) {
    double min=1;double max=6;
    attributes[0]=automated_analysis=setatt(rating, min, max);
}
void project::speer_reviews(double rating) {
    double min=1;double max=6;
    attributes[1]=peer_reviews=setatt(rating, min, max);
}
void project::sexecution_testing_and_tools(double rating) {
    double min=1;double max=6;
    attributes[2]=execution_testing_and_tools=setatt(rating, min, max);
}

void project::spCoqualRSlope(double val) {attributes[25]=pCoqualRSlope=val;}
void project::snCoqualRSlope(double val) {attributes[26]=nCoqualRSlope=val;}
void project::sdfrCoqualRSlope(double val) {attributes[27]=dfrCoqualRSlope=val;}

void project::spCoqualDSlope(double val) {attributes[28]=pCoqualDSlope=val;}
void project::snCoqualDSlope(double val) {attributes[29]=nCoqualDSlope=val;}
void project::sdfrCoqualDSlope(double val) {attributes[30]=dfrCoqualDSlope=val;}

void project::spCoqualCSlope(double val) {attributes[31]=pCoqualCSlope=val;}
void project::snCoqualCSlope(double val) {attributes[32]=nCoqualCSlope=val;}
void project::sdfrCoqualCSlope(double val) {attributes[33]=dfrCoqualCSlope=val;}

void project::sSFCocomoSlope(double val) {attributes[34]=SFCocomoSlope=val;}
void project::sposEMCocomoSlope(double val) {attributes[35]=posEMCocomoSlope=val;}
void project::snegEMCocomoSlope(double val) {attributes[36]=negEMCocomoSlope=val;}

void project::sA(double val) {
        if (val==0.0) {
            randomNum rdm;
            attributes[37]=A=((B-(1.075-rdm.randomGenerateDouble(0.0, 0.1)))/(-0.0385135));
            if(A<attrangeNorm[0][37]) attributes[37]=A=attrangeNorm[0][37];
            if(A>attrangeNorm[1][37]) attributes[37]=A=attrangeNorm[1][37];
        }
        else {
                attributes[37]=A=val; sB(0.0);
        }
}
void project::sB(double val) {
        if (val==0.0) {
            randomNum rdm;
            attributes[38]=B=(-0.0385135*A+(1.075-rdm.randomGenerateDouble(0.0, 0.1)));
            if(B<attrangeNorm[0][38]) attributes[38]=B=attrangeNorm[0][38];
            if(B>attrangeNorm[1][38]) attributes[38]=B=attrangeNorm[1][38];
        }
        else {
                attributes[38]=B=val; sA(0.0);
        }
}

void project::sKsloc(double val) {attributes[39]=ksloc=val;}

void project::sC(double val) {attributes[40]=C=val;}

void project::sD(double val) {attributes[41]=D=val;}

string project::displayAttName(int attnum) {
    switch (attnum) {
        case 0: return "automated_analysis"; break;
        case 1: return "peer_reviews"; break;
        case 2: return "execution_testing_and_tools"; break;
        case 3: return "prec"; break;
        case 4: return "flex"; break;
        case 5: return "resl"; break;
        case 6: return "team"; break;
        case 7: return "pmat"; break;
        case 8: return "time"; break;
        case 9: return "stor"; break;
        case 10: return "data"; break;
        case 11: return "pvol"; break;
        case 12: return "ruse"; break;
        case 13: return "rely"; break;
        case 14: return "docu"; break;
```

```
        case 15: return "acap"; break;
        case 16: return "pcap"; break;
        case 17: return "pcon"; break;
        case 18: return "apex"; break;
        case 19: return "plex"; break;
        case 20: return "ltex"; break;
        case 21: return "tool"; break;
        case 22: return "sced"; break;
        case 23: return "cplx"; break;
        case 24: return "site"; break;
        //Slopes, A, B and Ksloc
        case 25: return "pCoqualRSlope"; break;
        case 26: return "nCoqualRSlope"; break;
        case 27: return "dfrCoqualRSlope"; break;
        case 28: return "pCoqualDSlope"; break;
        case 29: return "nCoqualDSlope"; break;
        case 30: return "dfrCoqualDSlope"; break;
        case 31: return "pCoqualCSlope"; break;
        case 32: return "nCoqualCSlope"; break;
        case 33: return "dfrCoqualCSlope"; break;
        case 34: return "SFCocomoSlope"; break;
        case 35: return "posEMCocomoSlope"; break;
        case 36: return "negEMCocomoSlope"; break;
        case 37: return "A"; break;
        case 38: return "B"; break;
        case 39: return "Ksloc"; break;
        case 40: return "C"; break;
        case 41: return "D"; break;
        default: return ""; break;
    }
}

void project::displayAtt(int attnum) {
    switch (attnum) {
        case 0: dispautomated_analysis(); break;
        case 1: disppeer_reviews(); break;
        case 2: dispexecution_testing_and_tools(); break;
        case 3: dispprec(); break;
        case 4: dispflex(); break;
        case 5: dispresl(); break;
        case 6: dispteam(); break;
        case 7: disppmat(); break;
        case 8: disptime(); break;
        case 9: dispstor(); break;
        case 10: dispdata(); break;
        case 11: disppvol(); break;
        case 12: dispruse(); break;
        case 13: disprely(); break;
        case 14: dispdocu(); break;
        case 15: dispacap(); break;
        case 16: disppcap(); break;
        case 17: disppcon(); break;
        case 18: dispapex(); break;
        case 19: disppplex(); break;
        case 20: displtex(); break;
        case 21: disptool(); break;
        case 22: dispsced(); break;
        case 23: dispcplx(); break;
        case 24: dispsite(); break;
        //Slopes, A, B and Ksloc
        case 25: disppCoqualRSlope(); break;
        case 26: dispnCoqualRSlope();break;
        case 27: dispdfrCoqualRSlope();break;
        case 28: disppCoqualDSlope();break;
        case 29: dispnCoqualDSlope();break;
        case 30: dispdfrCoqualDSlope();break;
        case 31: disppCoqualCSlope();break;
        case 32: dispnCoqualCSlope();break;
        case 33: dispdfrCoqualCSlope();break;
        case 34: dispSFCocomoSlope();break;
        case 35: dispposEMCocomoSlope();break;
        case 36: dispnegEMCocomoSlope();break;
        case 37: dispA(); break;
        case 38: dispB(); break;
        case 39: dispKsloc(); break;
        case 40: dispC(); break;
        case 41: dispD(); break;
    }
}

void project::dispprec() {cout << "prec: " << prec << endl;}
void project::dispflex() {cout << "flex: " << flex << endl;}
void project::dispresl() {cout << "resl: " << resl << endl;}
void project::dispteam() {cout << "team: " << team << endl;}
void project::disppmat() {cout << "pmat: " << pmat << endl;}

void project::disptime() {cout << "time: " << time << endl;}
```

```
void project::dispstor() {cout << "stor: " << stor << endl;}
void project::dispdata() {cout << "data: " << data << endl;}
void project::disppvol() {cout << "pvol: " << pvol << endl;}
void project::dispruse() {cout << "ruse: " << ruse << endl;}
void project::disprely() {cout << "rely: " << rely << endl;}
void project::dispdocu() {cout << "docu: " << docu << endl;}
void project::dispacap() {cout << "acap: " << acap << endl;}
void project::disppcap() {cout << "pcap: " << pcap << endl;}
void project::disppcon() {cout << "pcon: " << pcon << endl;}
void project::dispapex() {cout << "apex: " << apex << endl;}
void project::displtex() {cout << "ltex: " << ltex << endl;}
void project::disptool() {cout << "tool: " << tool << endl;}
void project::dispsced() {cout << "sced: " << sced << endl;}
void project::dispcplx() {cout << "cplx: " << cplx << endl;}
void project::dispsite() {cout << "site: " << site << endl;}
void project::dispplex() {cout << "plex: " << plex << endl;}

void project::dispautomated_analysis() {cout << "automated_analysis: " << automated_analysis << endl;}
void project::disppeer_reviews() {cout << "peer_reviews: " << peer_reviews << endl;}
void project::dispexecution_testing_and_tools() {cout << "execution_testing_and_tools: "
       << execution_testing_and_tools << endl;}

void project::disppCoqualRSlope() {cout << "pCoqualRSlope: " << pCoqualRSlope << endl;}
void project::dispnCoqualRSlope() {cout << "nCoqualRSlope: " << nCoqualRSlope << endl;}
void project::dispdfrCoqualRSlope() {cout << "dfrCoqualRSlope: " << dfrCoqualRSlope << endl;}
void project::disppCoqualDSlope() {cout << "pCoqualDSlope: " << pCoqualDSlope << endl;}
void project::dispnCoqualDSlope() {cout << "nCoqualDSlope: " << nCoqualDSlope << endl;}
void project::dispdfrCoqualDSlope() {cout << "dfrCoqualDSlope: " << dfrCoqualDSlope << endl;}
void project::disppCoqualCSlope() {cout << "pCoqualCSlope: " << pCoqualCSlope << endl;}
void project::dispnCoqualCSlope() {cout << "nCoqualCSlope: " << nCoqualCSlope << endl;}
void project::dispdfrCoqualCSlope() {cout << "dfrCoqualCSlope: " << dfrCoqualCSlope << endl;}
void project::dispSFCocomoSlope() {cout << "SFCocomoSlope: " << SFCocomoSlope << endl;}
void project::dispposEMCocomoSlope() {cout << "posEMCocomoSlope: " << posEMCocomoSlope << endl;}
void project::dispnegEMCocomoSlope() {cout << "negEMCocomoSlope: " << negEMCocomoSlope << endl;}
void project::dispA() {cout << "A: " << A << endl;}
void project::dispB() {cout << "B: " << B << endl;}
void project::dispKsloc() {cout << "ksloc: " << ksloc << endl;}
void project::dispC() {cout << "C: " << C << endl;}
void project::dispD() {cout << "D: " << D << endl;}

double project::effort() {
    return A*(pow(ksloc, (B+0.01*(fprec()+fflex()+fresl()+fteam()+fpmat()))))
            *(ftime()*fstor()*fdata()*fpvol()*fruse()*frely()*fdocu()*facap()*fpcap()*
            fpcon()*fapex()*fplex()*fltex()*ftool()*fsced()*fcplx()*fsite());
}

//Time to develop Model
//
double project::pmNs(){
    return (A*pow(ksloc, (B+0.01*(fprec()+fflex()+fresl()+fteam()+fpmat())))
            *(ftime()*fstor()*fdata()*fpvol()*fruse()*frely()*fdocu()*facap()*fpcap()*
            fpcon()*fapex()*fplex()*fltex()*ftool()*fcplx()*fsite()));
}

double project::scedpercent(){
    switch((int)round(sced)){
        case 1: return(75);
        case 2: return(85);
        case 3: return(100);
        case 4: return(130);
        case 5: return(160);
    }
}


double project::months(){
    return ((C*pow(pmNs(), (D+0.2*(0.01*(fprec()+fflex()+fresl()+fteam()+fpmat()))))))*(scedpercent()/100));
}



//Threat model
//Threat model tables
double project::t1[5][5] = {{0, 0, 0, 1, 2},
                                                        {0, 0, 0, 0, 1}};
double project::t2[5][6] = {{0, 0, 0, 1, 2, 4},
                                                        {0, 0, 0, 0, 1, 2},
                                                        {0, 0, 0, 0, 0, 1}};
double project::t3[5][5] = {{2, 1},
                                                        {1}};
double project::t4[5][5] = {{4, 2, 1},
                                                        {2, 1},
                                                        {1}};
double project::t5[5][5] = {{0},
                                                        {0},
                                                        {1},
                                                        {2, 1},
```

```
                                                    {4, 2, 1}};
double project::t6[6][5] = {{0},
                                                    {0},
                                                    {0},
                                                    {1},
                                                    {2, 1},
                                                    {4, 2, 1}};
double project::t7[5][5] = {{0},
                                                    {0},
                                                    {0},
                                                    {1},
                                                    {2, 1}};
double project::t8[6][5] = {{0},
                                                    {0},
                                                    {0},
                                                    {0},
                                                    {1},
                                                    {2, 1}};

double project::threat() {
    double TH = (double)(sched_threat()+prod_threat()+pers_threat()+proc_threat()
    +plat_threat()+reuse_threat())/3.73;

    if (TH < 2.5) TH = 0.0000000000000001;
    return TH;
}

double project::sched_threat() {
    return (t1[(int)round(sced-1)][(int)round(rely-1)]*fsced()*frely()+t2[(int)round(sced-1)]
    [(int)round(time-1)]*fsced()*ftime()+t1[(int)round(sced-1)][(int)round(pvol-1)]*fsced()*fpvol()+
    t3[(int)round(sced-1)][(int)round(tool-1)]*fsced()*ftool()+t4[(int)round(sced-1)][(int)round(acap-1)]
    *fsced()*facap()+t4[(int)round(sced-1)][(int)round(apex-1)]*fsced()*fapex()+t4[(int)round(sced-1)]
    [(int)round(pcap-1)]*fsced()*fpcap()+t4[(int)round(sced-1)][(int)round(plex-1)]*fsced()*fplex()+
    t3[(int)round(sced-1)][(int)round(ltex-1)]*fsced()*fltex()+t3[(int)round(sced-1)][(int)round(pmat-1)]
    *fsced()*fpmat());
}

double project::prod_threat() {
    return (t5[(int)round(rely-1)][(int)round(acap-1)]*frely()*facap()+t5[(int)round(rely-1)]
    [(int)round(pcap-1)]*frely()*fpcap()+t6[(int)round(cplx-1)][(int)round(acap-1)]*fcplx()*facap()+
    t6[(int)round(cplx-1)][(int)round(pcap-1)]*fcplx()*fpcap()+t6[(int)round(cplx-1)][(int)round(tool-1)]
    *fcplx()*ftool()+t5[(int)round(rely-1)][(int)round(pmat-1)]*frely()*fpmat()+t2[(int)round(sced-1)]
    [(int)round(cplx-1)]*fsced()*fcplx()+t1[(int)round(sced-1)][(int)round(rely-1)]*fsced()*frely()+
    t2[(int)round(sced-1)][(int)round(time-1)]*fsced()*ftime()+t6[(int)round(ruse-1)][(int)round(apex-1)]
    *fruse()*fapex()+t6[(int)round(ruse-1)][(int)round(ltex-1)])*fruse()*fltex();
}

double project::pers_threat() {
    return (t3[(int)round(pmat-1)][(int)round(acap-1)]*fpmat()*facap()+t6[(int)round(stor-1)]
    [(int)round(acap-1)]*fstor()*facap()+t6[(int)round(time-1)][(int)round(acap-1)]*ftime()*facap()+
    t3[(int)round(tool-1)][(int)round(acap-1)]*ftool()*facap()+t3[(int)round(tool-1)][(int)round(pcap-1)]
    *ftool()*fpcap()+t6[(int)round(ruse-1)][(int)round(apex-1)]*fruse()*fapex()+t6[(int)round(ruse-1)]
    [(int)round(ltex-1)]*fruse()*fltex()+t3[(int)round(pmat-1)][(int)round(pcap-1)]*fpmat()*fpcap()+
    t6[(int)round(stor-1)][(int)round(pcap-1)]*fstor()*fpcap()+t6[(int)round(time-1)][(int)round(pcap-1)]
    *ftime()*fpcap()+t4[(int)round(ltex-1)][(int)round(pcap-1)]*fltex()*fpcap()+t7[(int)round(pvol-1)]
    [(int)round(plex-1)]*fpvol()*fplex()+t4[(int)round(sced-1)][(int)round(acap-1)]*fsced()*facap()+
    t4[(int)round(sced-1)][(int)round(apex-1)]*fsced()*fapex()+t4[(int)round(sced-1)][(int)round(pcap-1)]
    *fsced()*fpcap()+t4[(int)round(sced-1)][(int)round(plex-1)]*fsced()*fplex()+t3[(int)round(sced-1)]
    [(int)round(ltex-1)]*fsced()*fltex()+t5[(int)round(rely-1)][(int)round(acap-1)]*frely()*facap()+
    t5[(int)round(rely-1)][(int)round(pcap-1)]*frely()*fpcap()+t6[(int)round(cplx-1)][(int)round(acap-1)]
    *fcplx()*facap()+t6[(int)round(cplx-1)][(int)round(pcap-1)]*fcplx()*fpcap()+t3[(int)round(team-1)]
    [(int)round(apex-1)]*fteam()*fapex());
}

double project::proc_threat() {
    return (t3[(int)round(tool-1)][(int)round(pmat-1)]*ftool()*fpmat()+t8[(int)round(time-1)]
    [(int)round(tool-1)]*ftime()*ftool()+t3[(int)round(tool-1)][(int)round(pmat-1)]*ftool()*fpmat()+
    t3[(int)round(team-1)][(int)round(apex-1)]*fteam()*fapex()+t3[(int)round(team-1)][(int)round(sced-1)]
    *fteam()*fsced()+t3[(int)round(team-1)][(int)round(site-1)]*fteam()*fsite()+t3[(int)round(sced-1)]
    [(int)round(tool-1)]*fsced()*ftool()+t3[(int)round(sced-1)][(int)round(pmat-1)]*fsced()*fpmat()+
    t6[(int)round(cplx-1)][(int)round(tool-1)]*fcplx()*ftool()+t3[(int)round(pmat-1)][(int)round(acap-1)]
    *fpmat()*facap()+t3[(int)round(tool-1)][(int)round(acap-1)]*ftool()*facap()+t3[(int)round(tool-1)]
    [(int)round(pcap-1)]*ftool()*fpcap()+t3[(int)round(pmat-1)][(int)round(pcap-1)]*fpmat()*fpcap());
}

double project::plat_threat() {
    return (t2[(int)round(sced-1)][(int)round(time-1)]*fsced()*ftime()+t1[(int)round(sced-1)]
    [(int)round(pvol-1)]*fsced()*fpvol()+t6[(int)round(stor-1)][(int)round(acap-1)]*fstor()*facap()+
    t6[(int)round(time-1)][(int)round(acap-1)]*ftime()*facap()+t6[(int)round(stor-1)][(int)round(pcap-1)]
    *fstor()*fpcap()+t7[(int)round(pvol-1)][(int)round(plex-1)]*fpvol()*fplex()+t8[(int)round(time-1)]
    [(int)round(tool-1)]*ftime()*ftool());
}

double project::reuse_threat() {
    return (t6[(int)round(ruse-1)][(int)round(apex-1)]*fruse()*fapex()+t6[(int)round(ruse-1)]
    [(int)round(ltex-1)]*fruse()*fltex());
}
```

```
double project::posCoqualR(double rating) {return (double)(rating-3)*pCoqualRSlope + 1;}
double project::negCoqualR(double rating) {return (double)(rating-3)*nCoqualRSlope + 1;}
double project::dfrCoqualR(double rating) {return (double)(rating-1)*dfrCoqualRSlope;}

double project::posCoqualD(double rating) {return (double)(rating-3)*pCoqualDSlope + 1;}
double project::negCoqualD(double rating) {return (double)(rating-3)*nCoqualDSlope + 1;}
double project::dfrCoqualD(double rating) {return (double)(rating-1)*dfrCoqualDSlope;}

double project::posCoqualC(double rating) {return (double)(rating-3)*pCoqualCSlope + 1;}
double project::negCoqualC(double rating) {return (double)(rating-3)*nCoqualCSlope + 1;}
double project::dfrCoqualC(double rating) {return (double)(rating-1)*dfrCoqualCSlope;}


double project::flexR() {return 1.0;}
double project::pcapR() {return 1.0;}

double project::ruseR() {return posCoqualR(ruse);}
double project::cplxR() {return posCoqualR(cplx);}
double project::timeR() {return posCoqualR(time);}
double project::storR() {return posCoqualR(stor);}
double project::pvolR() {return posCoqualR(pvol);}
double project::dataR() {return posCoqualR(data);}

double project::acapR() {return negCoqualR(acap);}
double project::pconR() {return negCoqualR(pcon);}
double project::apexR() {return negCoqualR(apex);}
double project::plexR() {return negCoqualR(plex);}
double project::ltexR() {return negCoqualR(ltex);}
double project::toolR() {return negCoqualR(tool);}
double project::siteR() {return negCoqualR(site);}
double project::scedR() {return negCoqualR(sced);}
double project::relyR() {return negCoqualR(rely);}
double project::docuR() {return negCoqualR(docu);}
double project::reslR() {return negCoqualR(resl);}
double project::teamR() {return negCoqualR(team);}
double project::pmatR() {return negCoqualR(pmat);}
double project::precR() {return negCoqualR(prec);}

double project::flexD() {return 1.0;}

double project::ruseD() {return posCoqualD(ruse);}
double project::cplxD() {return posCoqualD(cplx);}
double project::timeD() {return posCoqualD(time);}
double project::storD() {return posCoqualD(stor);}
double project::pvolD() {return posCoqualD(pvol);}
double project::dataD() {return posCoqualD(data);}

double project::acapD() {return negCoqualD(acap);}
double project::pconD() {return negCoqualD(pcon);}
double project::apexD() {return negCoqualD(apex);}
double project::plexD() {return negCoqualD(plex);}
double project::ltexD() {return negCoqualD(ltex);}
double project::toolD() {return negCoqualD(tool);}
double project::siteD() {return negCoqualD(site);}
double project::scedD() {return negCoqualD(sced);}
double project::relyD() {return negCoqualD(rely);}
double project::docuD() {return negCoqualD(docu);}
double project::reslD() {return negCoqualD(resl);}
double project::teamD() {return negCoqualD(team);}
double project::pmatD() {return negCoqualD(pmat);}
double project::precD() {return negCoqualD(prec);}
double project::pcapD() {return negCoqualD(pcap);}

double project::flexC() {return 1.0;}

double project::ruseC() {return posCoqualC(ruse);}
double project::cplxC() {return posCoqualC(cplx);}
double project::timeC() {return posCoqualC(time);}
double project::storC() {return posCoqualC(stor);}
double project::pvolC() {return posCoqualC(pvol);}
double project::dataC() {return posCoqualC(data);}

double project::acapC() {return negCoqualC(acap);}
double project::pconC() {return negCoqualC(pcon);}
double project::apexC() {return negCoqualC(apex);}
double project::plexC() {return negCoqualC(plex);}
double project::ltexC() {return negCoqualC(ltex);}
double project::toolC() {return negCoqualC(tool);}
double project::siteC() {return negCoqualC(site);}
double project::scedC() {return negCoqualC(sced);}
double project::relyC() {return negCoqualC(rely);}
double project::docuC() {return negCoqualC(docu);}
double project::reslC() {return negCoqualC(resl);}
double project::teamC() {return negCoqualC(team);}
double project::pmatC() {return negCoqualC(pmat);}
```

```
double project::precC() {return negCoqualC(prec);}
double project::pcapC() {return negCoqualC(pcap);}

double project::automated_analysisR() {
//      switch (automated_analysis) {
//              case 1: return 0.0; break;
//              case 2: return 0.0; break;
//              case 3: return 0.1; break;
//              case 4: return 0.27; break;
//              case 5: return 0.34; break;
//              case 6: return 0.4; break;
//              default: std::cout << "automated_analysis out of range\n"; return 0.0; break;
//      }

    return dfrCoqualR(automated_analysis);
}
double project::peer_reviewsR() {
//      switch (peer_reviews) {
//              case 1: return 0.0; break;
//              case 2: return 0.25; break;
//              case 3: return 0.4; break;
//              case 4: return 0.5; break;
//              case 5: return 0.58; break;
//              case 6: return 0.7; break;
//              default: std::cout << "peer_reviews out of range\n"; return 0.0; break;
//      }

    return dfrCoqualR(peer_reviews);
}
double project::execution_testing_and_toolsR() {
    return dfrCoqualR(execution_testing_and_tools);
}

double project::automated_analysisD() {
    return dfrCoqualD(automated_analysis);
}
double project::peer_reviewsD() {
    return dfrCoqualD(peer_reviews);
}
double project::execution_testing_and_toolsD() {
    return dfrCoqualD(execution_testing_and_tools);
}

double project::automated_analysisC() {
    return dfrCoqualC(automated_analysis);
}
double project::peer_reviewsC() {
    return dfrCoqualC(peer_reviews);
}
double project::execution_testing_and_toolsC() {
    return dfrCoqualC(execution_testing_and_tools);
}

double project::defectsIntroR() {
    return flexR()*pcapR()*ruseR()*cplxR()*timeR()*storR()*pvolR()*dataR()
           *acapR()*pconR()*apexR()*plexR()*ltexR()*toolR()*siteR()*scedR()*relyR()
           *docuR()*reslR()*teamR()*pmatR()*precR();
}
double project::defectsIntroD() {
    return flexD()*pcapD()*ruseD()*cplxD()*timeD()*storD()*pvolD()*dataD()
           *acapD()*pconD()*apexD()*plexD()*ltexD()*toolD()*siteD()*scedD()*relyD()
           *docuD()*reslD()*teamD()*pmatD()*precD();
}
double project::defectsIntroC() {
    return flexC()*pcapC()*ruseC()*cplxC()*timeC()*storC()*pvolC()*dataC()
           *acapC()*pconC()*apexC()*plexC()*ltexC()*toolC()*siteC()*scedC()*relyC()
           *docuC()*reslC()*teamC()*pmatC()*precC();
}

double project::defectsRemR() {
    return (1-automated_analysisR())*(1-peer_reviewsR())*(1-execution_testing_and_toolsR());
}

double project::defectsRemD() {
    return (1-automated_analysisD())*(1-peer_reviewsD())*(1-execution_testing_and_toolsD());
}

double project::defectsRemC() {
    return (1-automated_analysisC())*(1-peer_reviewsC())*(1-execution_testing_and_toolsC());
}

double project::defects() {
    return ksloc*(10*defectsIntroR()*defectsRemR()
           +20*defectsIntroD()*defectsRemD()
           +30*defectsIntroC()*defectsRemC());
}
```

```
double project::threatMutate() {
    randomNum rdm;
    double RNum;

    double tmp1 [5][5];
    double tmp2 [5][6];
    double tmp3 [5][5];
    double tmp4 [5][5];
    double tmp5 [5][5];
    double tmp6 [6][5];
    double tmp7 [5][5];
    double tmp8 [6][5];

    bool mutate[8] = {false, false, false, false, false, false, false, false};

    for (int i=0;i<8;i++) {
        RNum = rdm.randomGenerateDouble(0, 1.5);
        if (RNum > 0.5) {
            mutate[i]=true;
        }
    }


    if (mutate[0]) {
        RNum = rdm.randomGenerateDouble(0.5, 1.5);
        for (int i=0;i<5;i++) {
            for (int j=0;j<5;j++) {
                tmp1[i][j]=t1[i][j];
                t1[i][j]*=RNum;
            }
        }
    }

    if (mutate[1]) {
        RNum = rdm.randomGenerateDouble(0.5, 1.5);
        for (int i=0;i<5;i++) {
            for (int j=0;j<6;j++) {
                tmp2[i][j]=t2[i][j];
                t2[i][j]*=RNum;
            }
        }
    }

    if (mutate[2]) {
        RNum = rdm.randomGenerateDouble(0.5, 1.5);
        for (int i=0;i<5;i++) {
            for (int j=0;j<5;j++) {
                tmp3[i][j]=t3[i][j];
                t3[i][j]*=RNum;
            }
        }
    }

    if (mutate[3]) {
        RNum = rdm.randomGenerateDouble(0.5, 1.5);
        for (int i=0;i<5;i++) {
            for (int j=0;j<5;j++) {
                tmp4[i][j]=t4[i][j];
                t4[i][j]*=RNum;
            }
        }
    }

    if (mutate[4]) {
        RNum = rdm.randomGenerateDouble(0.5, 1.5);
        for (int i=0;i<5;i++) {
            for (int j=0;j<5;j++) {
                tmp5[i][j]=t5[i][j];
                t5[i][j]*=RNum;
            }
        }
    }

    if (mutate[5]) {
        RNum = rdm.randomGenerateDouble(0.5, 1.5);
        for (int i=0;i<6;i++) {
            for (int j=0;j<5;j++) {
                tmp6[i][j]=t6[i][j];
                t6[i][j]*=RNum;
            }
        }
    }

    if (mutate[6]) {
        RNum = rdm.randomGenerateDouble(0.5, 1.5);
        for (int i=0;i<5;i++) {
            for (int j=0;j<5;j++) {
```

```
                tmp7[i][j]=t7[i][j];
                t7[i][j]*=RNum;
            }
        }
    }

    if (mutate[7]) {
        RNum = rdm.randomGenerateDouble(0.5, 1.5);
        for (int i=0;i<6;i++) {
            for (int j=0;j<5;j++) {
                tmp8[i][j]=t8[i][j];
                t8[i][j]*=RNum;
            }
        }
    }

    double TH = threat();


    if (mutate[0]) {
        for (int i=0;i<5;i++) {
            for (int j=0;j<5;j++) {
                t1[i][j]=tmp1[i][j];
            }
        }

    }

    if (mutate[1]) {
        for (int i=0;i<5;i++) {
            for (int j=0;j<6;j++) {
                t2[i][j]=tmp2[i][j];
            }
        }
    }

    if (mutate[2]) {
        for (int i=0;i<5;i++) {
            for (int j=0;j<5;j++) {
                t3[i][j]=tmp3[i][j];
            }
        }
    }

    if (mutate[3]) {
        for (int i=0;i<5;i++) {
            for (int j=0;j<5;j++) {
                t4[i][j]=tmp4[i][j];
            }
        }
    }

    if (mutate[4]) {
        for (int i=0;i<5;i++) {
            for (int j=0;j<5;j++) {
                t5[i][j]=tmp5[i][j];
            }
        }
    }

    if (mutate[5]) {
        for (int i=0;i<6;i++) {
            for (int j=0;j<5;j++) {
                t6[i][j]=tmp6[i][j];
            }
        }
    }

    if (mutate[6]) {
        for (int i=0;i<5;i++) {
            for (int j=0;j<5;j++) {
                t7[i][j]=tmp7[i][j];
            }
        }
    }

    if (mutate[7]) {
        for (int i=0;i<6;i++) {
            for (int j=0;j<5;j++) {
                t8[i][j]=tmp8[i][j];
            }
        }
    }

    return TH;
}
```

```
double project::effortMutate() {

    project s = *this;

    double AttV [22] = {0};

    randomNum rdm;

    for (int i=0; i<22; i++) {
        for (int l=34; l<37; l++) {
            s.setattnum(l, rdm.randomGenerateDouble(s.attrangeNorm[0][l], s.attrangeNorm[1][l]));
        }

        AttV [i] = s.getattnumvaleff(i+1);
    }

    double SF = 0;

    for (int i=0; i<5; i++) {
        SF += AttV[i];
    }

    double EM = 1;

    for (int i=5; i<22; i++) {
        EM *= AttV[i];
    }

    double eff = 0;
    eff = s.A*(pow(s.ksloc, (s.B+(0.01*SF))))*EM;

    return eff;
}

double project::defectsMutate() {
    project s= *this;

    double AttVR [25] = {0};
    double AttVD [25] = {0};
    double AttVC [25] = {0};

    randomNum rdm;

    for (int i=0; i<25; i++) {
        for (int l=25; l<28; l++) {
            s.setattnum(l, rdm.randomGenerateDouble(s.attrangeNorm[0][l], s.attrangeNorm[1][l]));
        }

        AttVR [i] = s.getattnumvaldefR(i);
    }

    for (int i=0; i<25; i++) {
        for (int l=28; l<31; l++) {
            s.setattnum(l, rdm.randomGenerateDouble(s.attrangeNorm[0][l], s.attrangeNorm[1][l]));
        }

        AttVD [i] = s.getattnumvaldefD(i);
    }

    for (int i=0; i<25; i++) {
        for (int l=31; l<34; l++) {
            s.setattnum(l, rdm.randomGenerateDouble(s.attrangeNorm[0][l], s.attrangeNorm[1][l]));
        }

        AttVC [i] = s.getattnumvaldefC(i);
    }

    double defectsIntroR = 1;

    for (int i=3; i<25; i++) {
        defectsIntroR *= AttVR [i];
    }

    double defectsIntroD = 1;

    for (int i=3; i<25; i++) {
        defectsIntroD *= AttVD [i];
    }

    double defectsIntroC = 1;

    for (int i=3; i<25; i++) {
        defectsIntroC *= AttVC [i];
    }

    double defectsRemR = 1;
```

```
    for (int i=0; i<3; i++) {
        defectsRemR *= (1-AttVR[i]);
    }

    double defectsRemD = 1;

    for (int i=0; i<3; i++) {
        defectsRemD *= (1-AttVD[i]);
    }

    double defectsRemC = 1;

    for (int i=0; i<3; i++) {
        defectsRemC *= (1-AttVC[i]);
    }

    double def = 0;
    def = s.ksloc*((10*defectsIntroR*defectsRemR )+ (20*defectsIntroD*defectsRemD) +
     (30*defectsIntroC*defectsRemC));

    return def;
}

double project::monthsMutate() {
    project s= *this;

    double AttV[22] = {0};

    randomNum rdm;

    for (int i=0; i<22; i++) {
        if (i!=19) { //excluding sced
            for (int l=34; l<37; l++) {
                s.setattnum(l, rdm.randomGenerateDouble(s.attrangeNorm[0][l], s.attrangeNorm[1][l]));
            }

            AttV[i] = s.getattnumvaleff(i+1);
        }
    }

    double SF = 0;

    for (int i=0; i<5; i++) {
        SF += AttV[i];
    }

    double EMnS = 1;

    for (int i=5; i<22; i++) {
        if (i!=19) EMnS *= AttV[i]; //remove sced
    }

    double pmNs = s.A*(pow(s.ksloc, (s.B+(0.01*SF))))*EMnS;

    double mon = 0;
    mon = (s.C*(pow(pmNs, (s.D+0.2*(0.01*SF))))*(s.scedpercent()/100));

    return mon;
}
```

B.3 minmax.cpp

```
#include "minmax.h"
#include <fstream>

minmax::minmax() {
}

minmax::~minmax() {
}

double minmax::effortMax(project x) {

    project s=x;

    double maxEff=0;

    for (int i=0; i<x.ATTN; i++) {
        if (s.attributes[i]==0) {
            if((i>=8 && i<=14) || i==23 || i==35 || (i>=37 && i<=39))
                s.setattnum(i, s.attrangeNorm[1][i]);
            else
                s.setattnum(i, s.attrangeNorm[0][i]);
        }
```

```
    }

    s.B = s.attrangeNorm[1][38];

    maxEff = s.effort();

    return maxEff + 0.0000000000000001;
}

double minmax::effortMin(project x) {

    project s=x;

    double minEff=0;

    for (int i=0; i<x.ATTN; i++) {
        if (s.attributes[i]==0) {
            if((i>=8 && i<=14) || i==23 || i==34 || i==36 || (i>=37 && i<=39))
                s.setattnum(i, s.attrangeNorm[0][i]);
            else
                s.setattnum(i, s.attrangeNorm[1][i]);
        }
    }

    s.B = s.attrangeNorm[0][38];

    minEff = s.effort();

    return minEff;
}



////////////////////////////////////////////////////////////////////////////
// min max functions for monthsmodel
////////////////////////////////////////////////////////////////////////////

double minmax::monthsMax(project x) {

    project s=x;

    double maxMonths=0;

    for (int i=0; i<x.ATTN; i++) {
        if (s.attributes[i]==0) {
            if((i>=8 && i<=14) || i==23 || i==35 || (i>=37 && i<=41))
                s.setattnum(i, s.attrangeNorm[1][i]);
            else
                s.setattnum(i, s.attrangeNorm[0][i]);
        }
    }

    s.B = s.attrangeNorm[1][38];

    maxMonths = s.months();

    return maxMonths + 0.0000000000000001;
}


double minmax::monthsMin(project x) {

    project s=x;

    double minMonths=0;

    for (int i=0; i<x.ATTN; i++) {
        if (s.attributes[i]==0) {
            if((i>=8 && i<=14) || i==23 || i==34 || i==36 || (i>=37 && i<=41))
                s.setattnum(i, s.attrangeNorm[0][i]);
            else
                s.setattnum(i, s.attrangeNorm[1][i]);
        }
    }

    s.B = s.attrangeNorm[0][38];

    minMonths = s.months();

    return minMonths;
}


double minmax::defectMax(project x) {

    project s=x;
```

```
    double maxDef=0;

    for (int i=0; i<x.ATTN; i++) {
        if (s.attributes[i]==0) {
            if((i>=8 && i<=12) || i==23 || i==25 || i==27 || i==28 || i==30 || i==31 || i==33 || i==39)
                s.setattnum(i, s.attrangeNorm[1][i]);
            else
                s.setattnum(i, s.attrangeNorm[0][i]);
        }
    }

    maxDef = s.defects();

    return maxDef + 0.0000000000000001;
}

double minmax::defectMin(project x) {

    project s=x;

    double minDef=0;

    for (int i=0; i<x.ATTN; i++) {
        if (s.attributes[i]==0) {
            if((i>=8 && i<=12) || i==23 || i==26 || i==29 || i==32 || i==39)
                s.setattnum(i, s.attrangeNorm[0][i]);
            else
                s.setattnum(i, s.attrangeNorm[1][i]);
        }
    }

    minDef = s.defects();

    return minDef;
}

double minmax::thrMax(project x) {

    project s=x;

    for (int i=0;i<5;i++) {
        for (int j=0;j<5;j++) {
            s.t1[i][j]*=1.5;
            s.t3[i][j]*=1.5;
            s.t4[i][j]*=1.5;
            s.t5[i][j]*=1.5;
            s.t7[i][j]*=1.5;
        }
    }

    for (int i=0;i<5;i++) {
        for (int j=0;j<6;j++) {
            s.t2[i][j]*=1.5;
        }
    }

    for (int i=0;i<6;i++) {
        for (int j=0;j<5;j++) {
            s.t6[i][j]*=1.5;
            s.t8[i][j]*=1.5;
        }
    }

    for (int i=0; i<x.ATTN; i++) {
        if (s.attributes[i]==0) {
            if((i>=8 && i<=14) || i==23 || i==35)
                s.setattnum(i, s.attrangeNorm[1][i]);
            else
                s.setattnum(i, s.attrangeNorm[0][i]);
        }
    }

//      double test1 = s.sched_threat();
//      double test2 = s.prod_threat();
//      double test3 = s.pers_threat();
//      double test4 = s.proc_threat();
//      double test5 = s.plat_threat();
//      double test6 = s.reuse_threat();

    double TH = s.threat();

    return TH;
}

double minmax::thrMin(project x) {

    project s=x;
```

```
    for (int i=0;i<5;i++) {
        for (int j=0;j<5;j++) {
            s.t1[i][j]*=0.5;
            s.t3[i][j]*=0.5;
            s.t4[i][j]*=0.5;
            s.t5[i][j]*=0.5;
            s.t7[i][j]*=0.5;
        }
    }

    for (int i=0;i<5;i++) {
        for (int j=0;j<6;j++) {
            s.t2[i][j]*=0.5;
        }
    }

    for (int i=0;i<6;i++) {
        for (int j=0;j<5;j++) {
            s.t6[i][j]*=0.5;
            s.t8[i][j]*=0.5;
        }
    }

    for (int i=0; i<x.ATTN; i++) {
        if (s.attributes[i]==0) {
            if((i>=8 && i<=14) || i==23 || i==34 || i==36)
                s.setattnum(i, s.attrangeNorm[0][i]);
            else
                s.setattnum(i, s.attrangeNorm[1][i]);
        }
    }

    double TH = s.threat();

    return TH;

}
```

B.4 policy.awk

```
BEGIN {
        C[95]=1.96;
        C[99]=2.58;
        FS = ",";
        OFMT = "%.10f";
        CONVFMT = "%.10f";
        #file = "simlog.csv";
        min=1000000000000;
        minPoint=0;
        out;
        N;
        minEnergy;
        scoring;
        SANum;
        TotalTime;
        SATime;
        policyVal;
        projectVal;
        run;
        alpha;
        beta;
        gamma;
        delta;
        relydefect;
        coolFactor;
        stratVal;
        bfcVal;
        slog;
        simNum=0;
        getline;
        S[simNum]=0;
        S2[simNum]=0;
}

{
        policies[simNum] = $1 " = " $2;
        attributeName[simNum] = $1;

        if ($6<min) {min = $6; minPoint=simNum;}

        policyMedE[simNum] = $6;
        policySpE[simNum] = $7;
        policyMedEff[simNum] = $10;
```

```
        policySpEff[simNum] = $11;
        policyMedDef[simNum] = $14;
        policySpDef[simNum] = $15;
        policyMedTh[simNum] = $18;
        policySpTh[simNum] = $19;
        policyMedMon[simNum] = $22;
        policySpMon[simNum] = $23;

        simNum++;

        S[simNum]=$4;
        S2[simNum]=$5;
}
END {
        for (i=minPoint; i<simNum-1; i++) {
                for (j=minPoint; j<simNum-1; j++) {
                        polEval[i,j] = compare(95,S[i],S[j],S2[i],S2[j],N,N);
                }
        }

        for (j=minPoint; j<simNum-1; j++) {
                if (polEval[minPoint,j] == "=" || polEval[minPoint,j] == "<" ) minPoint = j;
        }


        numPol = 0;

        if (slog==1) print "\nNumber of total policies is " simNum "\n";
        output = "Number of total policies is " simNum "\n\n";

        FirstMedE = 0;
        FirstSpE = 0;
        FirstMedEffort = 0;
        FirstSpEffort = 0;
        FirstMedDefects = 0;
        FirstSpDefects = 0;
        FirstMedThreat = 0;
        FirstSpThreat = 0;
        FirstMedMonths = 0;
        FirstSpMonths = 0;

        MinMedE = 0;
        MinSpE = 0;
        MinMedEffort = 0;
        MinSpEffort = 0;
        MinMedDefects = 0;
        MinSpDefects = 0;
        MinMedThreat = 0;
        MinSpThreat = 0;
        MinMedMonths = 0;
        MinSpMonths = 0;

        for (i=simNum-1; i>=minPoint; i--) {
                if (i==simNum-1) {
                        FirstMedE = policyMedE[i];
                        FirstSpE = policySpE[i];
                        FirstMedEffort = policyMedEff[i];
                        FirstSpEffort = policySpEff[i];
                        FirstMedDefects = policyMedDef[i];
                        FirstSpDefects = policySpDef[i];
                        FirstMedThreat = policyMedTh[i];
                        FirstSpThreat = policySpTh[i];
                        FirstMedMonths = policyMedMon[i];
                        FirstSpMonths = policySpMon[i];
                }

                if (slog==1) print policies[i];
                output = output policies[i] "\n";
                numPol++;

                atts[attributeName[i]]++;

                if (i == minPoint) {
                        MinMedE = policyMedE[i];
                        MinSpE = policySpE[i];
                        MinMedEffort = policyMedEff[i];
                        MinSpEffort = policySpEff[i];
                        MinMedDefects = policyMedDef[i];
                        MinSpDefects = policySpDef[i];
                        MinMedThreat = policyMedTh[i];
                        MinSpThreat = policySpTh[i];
                        MinMedMonths = policyMedMon[i];
                        MinSpMonths = policySpMon[i];

                        attNumber=0;

                        for (a in atts) {
```

```
                        attNumber++;
                }

                if (slog==1){
                print "\nNumber of recommended policies is " numPol;
                print "\nNumber of distinct attributes is " attNumber;
                print "\nResults of applying above policies:";
                print "\tEnergy:\n\t\tmedian: " policyMedE[i] "\n\t\tspread: " policySpE[i];
                print "\tEffort:\n\t\tmedian: " policyMedEff[i] "\n\t\tspread: " policySpEff[i];
                print "\tDefects:\n\t\tmedian: " policyMedDef[i] "\n\t\tspread: " policySpDef[i];
                print "\tThreat:\n\t\tmedian: " policyMedTh[i] "\n\t\tspread: " policySpTh[i];
                print "\tMonths:\n\t\tmedian: " policyMedMon[i] "\n\t\tspread: " policySpMon[i];
                }

                output = output "\nNumber of recommended policies is " numPol "\n";
                output = output "\nNumber of distinct attributes is " attNumber "\n";
                output = output "\nResults of applying above policies:";
                output = output "\n\tEnergy:\n\t\tmedian: " policyMedE[i] "\n\t\tspread: "
                 policySpE[i];
                output = output "\n\tEffort:\n\t\tmedian: " policyMedEff[i] "\n\t\tspread: "
                 policySpEff[i];
                output = output "\n\tDefects:\n\t\tmedian: " policyMedDef[i] "\n\t\tspread: "
                 policySpDef[i];
                output = output "\n\tThreat:\n\t\tmedian: " policyMedTh[i] "\n\t\tspread: "
                 policySpTh[i];
                output = output "\n\tMonths:\n\t\tmedian: " policyMedMon[i] "\n\t\tspread: "
                 policySpMon[i];

                break;
            }
        }
        outlog = out "/policy.txt";
        print output > outlog;

        outdata = out "/data";
        print projectVal "," policyVal "," scoring "," mutation "," alpha "," beta "," gamma "," delta ","\
            relydefect "," coolFactor "," run "," N "," SANum "," SATime "," TotalTime "," minEnergy \
            "," numPol "," attNumber "," FirstMedE "," FirstSpE "," FirstMedEffort "," FirstSpEffort \
            "," FirstMedDefects "," FirstSpDefects "," FirstMedThreat "," FirstSpThreat "," \
            FirstMedMonths "," FirstSpMonths "," MinMedE "," MinSpE "," MinMedEffort "," MinSpEffort \
            "," MinMedDefects "," MinSpDefects "," MinMedThreat "," MinSpThreat "," MinMedMonths "," \
            MinSpMonths "," stratVal "," bfcVal > outdata;
}

function compare(conf,befores,afters,befores2, afters2,beforen,aftern) {
        if(same(conf,befores,afters,befores2, afters2,beforen,aftern))
                return "=";
        if (befores/beforen > afters/aftern)
                return "<";
        else
                return ">";
}

function same(conf,befores,afters,befores2, afters2,beforen,aftern,\
        ssa,ssb,pooled,sxaxb,t,tcrtical) {

        ssa = befores2-(befores^2/beforen);
        ssb = afters2-(afters^2 /aftern);
        pooled = (ssa + ssb) /(beforen + aftern - 2);
        sxaxb = sqrt(pooled *(1/beforen + 1/aftern));

        t= (afters/aftern - befores/beforen)/sxaxb;
        t = (t < 0 ? -1*t : t)
        tcritical= 2.101; #C[conf]

        return tcritical > t;
}
```

Bibliography

[1] Beagle 2 mission profile. http://solarsystem.nasa.gov/missions/profile.cfm?MCode=Beagle_02.

[2] J. Aguilar-Ruiz, I. Ramos, J.C. Riquelme, and M. Toro. An evolutionary approach to estimating software development projects. *Information and Software Technology*, 43(14):875–882, December 2001.

[3] G. Antoniol, M. Di Penta, and M. Harman. Search-based techniques applied to optimization of project planning for a massive maintenance project. *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 240–249, Sept. 2005.

[4] Giuliano Antoniol, Massimiliano Di Penta, and Mark Harman. A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty. *Software Metrics, IEEE International Symposium on*, 0:172–183, 2004.

[5] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley. The next release problem. *Information and Software Technology*, 43(14):883 – 890, 2001.

[6] Dan Baker. A hybrid approach to expert and model-based effort estimation. Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007. Available from https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443.

[7] J. Bansiya and C.G. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, 28(1):4–17, 2002.

[8] Andr E Baresel, Daimlerchrysler Ag, Mark Harman, David Binkley, and Bogdan Korel. Evolutionary testing in the presence of loop-assigned flags: A testability transformation approach. In *In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2004*, pages 43–52. ACM, 2004.

[9] S.B. Bay and M.J. Pazzani. Detecting change in categorical data: Mining contrast sets. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, 1999. Available from http://www.ics.uci.edu/~pazzani/Publications/stucco.pdf.

[10] B.Boehm and H.In. Conflict analysis and negotiation aids for cost-quality requirements. *Software Quality Professional*, 1(2):38–50, March 1999.

[11] B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[12] B. Boehm. Safe and simple software cost analysis. *IEEE Software*, pages 14–17, September/October 2000. Available from http://www.computer.org/certification/beta/Boehm_Safe.pdf.

[13] B. Boehm, C. Abts, and S. Chulani. Software development cost estimation approaches - a survey. *Annals of Software Engineering*, 10:177–205, 2000.

[14] B. Boehm and P. Papaccio. Understanding and controlling software costs. *IEEE Trans. on Software Engineering*, 14(10):1462–1477, October 1988.

[15] Barry Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, and Chris Abts. *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[16] Barry Boehm and Hoh In. Conflict analysis and negotiation aids for cost-quality requirements, 1999. http://sunset.usc.edu/publications/TECHRPTS/1999/usccse1999-530/usccse1999-530.pdf.

[17] Salah Bouktif, Houari Sahraoui, and Giuliano Antoniol. Simulated annealing for improving software quality prediction. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1893–1900, New York, NY, USA, 2006. ACM.

[18] Lionel C. Briand, Yvan Labiche, and Marwa Shousha. Stress testing real-time systems with genetic algorithms. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1021–1028, New York, NY, USA, 2005. ACM.

[19] Zhihoa Chen, Tim Menzies, and Dan Port. Feature subset selection can improve software cost estimation. In *PROMISE'05*, 2005. Available from http://menzies.us/pdf/05/fsscocomo.pdf.

[20] S. Chulani, B. Boehm, and B. Steece. Bayesian analysis of empirical software engineering cost models. *IEEE Transaction on Software Engineerining*, 25(4), July/August 1999.

[21] J. Clark, J. J. Dolado, M. Harman, R. M. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd. Reformulating software engineering as a search problem. *IEE Proceedings on Software*, 150(3):161–175, 2003. Available from http://www.brunel.ac.uk/~csstrmh/papers/sbse.ps.

[22] R. Clark. Faster treatment learning, Computer Science, Portland State University. Master's thesis, 2005.

[23] Myra Cohen, Shiu Beng Kooi, and Witawas Srisa-an. Clustering the heap in multi-threaded applications for improved garbage collection. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1901–1908, New York, NY, USA, 2006. ACM.

[24] S. L. Cornford, M. S. Feather, J.R. Dunphy, J. Salcedo, and T. Menzies. Optimizing spacecraft design optimization engine development: Progress and plans. In *Proceedings of the IEEE Aerospace Conference, Big Sky, Montana*, 2003. Available from `http://menzies.us/pdf/03aero.pdf`.

[25] Keith Cowig. Nasa responds to the columbia accident report: Farewell to faster - better - cheaper, September 2003. http://www.spaceref.com/news/viewnews.html?id=864.

[26] J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*, 1994.

[27] Leonard David. Nasa report: Too many failures with faster, better, cheaper, March 2000. `http://www.space.com/businesstechnology/business/spear_report_000313.html`.

[28] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, Apr 2002.

[29] Tom DeMarco and Timothy Lister. *Peopleware: productive projects and teams*. Dorset House Publishing Co., Inc., New York, NY, USA, 1987.

[30] Sunita Devnani-Chulani. *Bayesian Analysis of Software Cost and Quality Models*. PhD thesis, 1999. Available on-line at `http://citeseer.ist.psu.edu/devnani-chulani99bayesian.html`.

[31] Pedro Domingos and Michael J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[32] M.S. Feather and T. Menzies. Converging on the optimal attainment of requirements. In *IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02, 9-13th September, University of Essen, Germany*, 2002. Available from `http://menzies.us/pdf/02re02.pdf`.

[33] Fred Glover and M. Laguna. Tabu search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, Oxford, England, 1993. Blackwell Scientific Publishing.

[34] Phillip Green. Impact of value-based software engineering on software process control. Master's thesis, WVU LCSEE dept., 2009.

[35] Omid Jalali Martin Feather Gregory Gay, Tim Menzies and James Kiper. Real-time optimization of requirements models. 2008. to be published.

[36] Donald Gross and Carl M. Harris. *Fundamentals of queueing theory (2nd ed.).* John Wiley & Sons, Inc., New York, NY, USA, 1985.

[37] M.A. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions On Knowledge And Data Engineering*, 15(6):1437– 1447, 2003. Available from `http://www.cs.waikato.ac.nz/~mhall/HallHolmesTKDE.pdf`.

[38] Mary hardin. Mars climate orbiter nearing sept. 23 arrival, September 1999. JPL Universe, Vol. 29, No. 19.

[39] M. Harman and B.F. Jones. Search-based software engineering. *Journal of Information and Software Technology*, 43:833–839, December 2001.

[40] Mark Harman. The current state and future of search based software engineering. pages 342–357, 2007.

[41] Mark Harman, Lin Hu, Rob Hierons, Joachim Wegener, Harmen Sthamer, Andr? Baresel, and Marc Roper. Testability transformation. *IEEE Transactions on Software Engineering*, 30(1):3–16, 2004.

[42] K. Havelund and T. Pressburger. Model checking java programs using java pathfinder. *International Journal on Software Tools for Technology Transfer*, 2(4), April 2000. Available from `http://ase.arc.nasa.gov/visser/jpf/jpf1.ps.gz`.

[43] John H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.

[44] G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[45] Y. Hu. Treatment learning, 2002. Masters thesis, Unviersity of British Columbia, Department of Electrical and Computer Engineering. In preperation.

[46] J. Huang and C. Ling. Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 17(3), March 2005.

[47] Harmen Sthamer Joachim Wegener, Andre Baresel. Evolutionary test environment for automatic structural testing. *Information and Software Technology*, 43(14):841–854, 2001.

[48] H. Kautz, B. Selman, and Y. Jiang. A general stochastic approach to solving problems with hard and soft constraints. In D. Gu, J. Du, and P. Pardalos, editors, *The Satisfiability Problem: Theory and Applications, New York, NY*, pages 573–586, 1997. Available on-line at `http://citeseer.ist.psu.edu/168907.html`.

[49] C.F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, May 1987.

[50] Sugarloaf Key. Columbia, the legacy of "better, faster, cheaper"?, July 2003. http://www.space-travel.com/reports/Columbia__The_Legacy_Of_Better_ _Faster__Cheaper.html.

[51] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[52] Colin Kirsopp, Martin J. Shepperd, and John Hart. Search heuristics, case-based reasoning and software project effort prediction. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1367–1374, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[53] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[54] J. Li and G. Ruhe. Decision support analysis for software effort estimation by analogy. In *Proceedings, PROMISE'07 workshop on Repeatable Experiments in Software Engineering*, 2007.

[55] R. Madachy. Knowledge-based risk assessment and cost estimation. In *Proceedings Ninth Knowledge-Based Software Engineering Conference*, pages 172 –178, 1994.

[56] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist.*, 18(1):50–60, 1947. Available on-line at http://projecteuclid.org/DPubS?service=UI&version=1.0&verb=Display&handle=euclid.aoms/1177730491.

[57] T. Menzies, D.Owen, and J. Richardson. The strangest thing about software. *IEEE Computer*, 2007. http://menzies.us/pdf/07strange.pdf.

[58] T. Menzies, O. Elrawas, D. Baker, J. Hihn, and K. Lum. On the value of stochastic abduction (if you fix everything, you lose fixes for everything else). In *International Workshop on Living with Uncertainty (an ASE'07 co-located event)*, 2007. Available from http://menzies.us/pdf/07fix.pdf.

[59] T. Menzies, O. Elrawas, B. Barry, R. Madachy, J. Hihn, D. Baker, and K. Lum. Accurate estimates without calibration. In *International Conference on Software Process*, 2008. Available from http://menzies.us/pdf/08icsp.pdf.

[60] T. Menzies, O. Elrawas, J. Hihn, M. Feathear, B. Boehm, and R. Madachy. The business case for automated software engineerng. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 303–312, New York, NY, USA, 2007. ACM.

[61] T. Menzies and Y. Hu. Data mining for very busy people. In *IEEE Computer*, November 2003. Available from http://menzies.us/pdf/03tar2.pdf.

[62] T. Menzies and J. Richardson. Xomo: Understanding development options for autonomy. In *COCOMO forum, 2005*, 2005. Available from http://menzies.us/pdf/05xomo_cocomo_forum.pdf. For more details, see also the longer technical report http://menzies.us/pdf/05xomo101.pdf.

[63] Tim Menzies, Zhihao Chen, Jairus Hihn, and Karen Lum. Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering*, November 2006. Available from http://menzies.us/pdf/06coseekmo.pdf.

[64] Tim Menzies, Steve Williams, Oussama ElRawas, Barry Boehm, and Jairus Hihn. How to avoid drastic software process change (using stochastic stability). In *International Conference on Software Engineering*, 2009. to be published in ICSE'09.

[65] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys*, 21:1087–1092, 1953.

[66] Brian S. Mitchell and Spiros Mancoridis. On the automatic modularization of software systems using the bunch tool. *IEEE Transactions on Software Engineering*, 32(3):193–208, 2006.

[67] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. (reprinted 1997,2000).

[68] NASA. Mars climate orbiter mishap investigation board phase i report. November 1999.

[69] Timeline of Faster Better Cheaper NASA watch. Faster - better - cheaper under fire. http://www.nasawatch.com/fbc.html.

[70] Mark O'Keeffe and Mel O'Cinneide. Search-based software maintenance. *Software Maintenance and Reengineering, European Conference on*, 0:249–260, 2006.

[71] Andres Orrego. The value of reuse in software process control. suggested paper for ICSP'09.

[72] H. Pohlheim. Genetic and evolutionary algorithm toolbox for use with matlab.

[73] INTERNATIONAL FEDERATION OF PROFESSIONAL and AFL-CIO TECHNICAL ENGINEERS. Ifpte report on the effectiveness of nasa's workforce & contractor policies, March 2003. http://www.spaceref.com/news/viewsr.html?pid=10275.

[74] Donald J. Reifer, Reifer Consultants, Barry W. Boehm, and Sunita Chulani. The rosetta stone: Making cocomo 81 estimates work with cocomo ii. 1999.

[75] Tony Spear. Nasa fbc task final report, March 2000. mars.jpl.nasa.gov/msp98/misc/fbctask.pdf.

[76] Tony Spear. Testimony on nasa fbc task before the subcommittee on science, technology, and space, March 2000. www.nasawatch.com/congress/2000/03.22.00.spear.pdf.

[77] Don Tuite. Better, faster, cheaperpick any two: That old mantra used to be a touchstone for development. but does it still ring true?, March 2007. http://electronicdesign.com/Articles/Index.cfm?AD=1&ArticleID=14997.

[78] Michael Turner. Faster, cheaper, and more ... metric?, August 2003. http://www.spacedaily.com/news/oped-03zz.html.

[79] R. Williams, C.P. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of IJCAI 2003*, 2003. http://www.cs.cornell.edu/gomes/FILES/backdoors.pdf.

[80] T. Young, J. Arnold, T. Brackey, M. Carr, D. Dwoyer, R. Fogleman, R. Jacobson, H. Kottler, P. Lyman, and J. Maguire. Mars program independent assessment team report. *NASA STI/Recon Technical Report N*, pages 32462–+, March 2000.

[81] H. Zhang and X. Zhang. Comments on 'data mining static code attributes to learn defect predictors'. *IEEE Transactions on Software Engineering*, September 2007.