On the Relative Merits of Software Reuse

Andres Orrego[1,2], Tim Menzies[2], and Oussama El-Rawas[2]

[1] Global Science & Technology, Inc., Fairmont, WV, USA
[2] West Virginia University, Morgantown, WV, USA.
andres.orrego@gst.com, tim@menzies.us, oelrawas@mix.wvu.edu

Abstract. Using process simulation and AI search methods, we compare software reuse against other possible changes to a project. such as reducing functionality or improving the skills of the programmer population. In one case, two generations of reuse were as good or better than any other project change (but a third and fourth generation of reuse was not useful). In another case, applying reuse to a project was demonstrable worse than several other possible changes to a project.
Our conclusion is that the general claims regarding the benefits of software reuse do not hold for specific projects. We argue that the merits of software reuse need to be evaluated in a project by project basis. AI search over process models is useful for such an assessment, particularly when there is not sufficient data for precisely tuning a simulation model.
Keywords: Software Reuse, COCOMO, COQUALMO, AI search

1 Introduction

We need to better understand software reuse. In theory, reuse can lower development cost, increase productivity, improve maintainability, boost quality, reduce risk, shorten life cycle time, lower training costs, and achieve better software interoperability [1, 2]. However, in practice, studies have shown that reuse is not always the best choice: it may be hard to implement, and the benefits of reuse cannot be reliably quantified [1]. Also, in some cases, reuse has resulted in economic loses [3] and even personal injury and loss of life [4].

Process simulations can be used to assess the value of reuse in a particular project. Traditionally, such simulators are commissioned using using data collected from a particular organization (e.g. [5]). Often, such local data is hard to collect. Accordingly, we have been been exploring an AI method called STAR that reduce the need for calibration from local data. To understand STAR, note that project estimates are some function of the project options and the internal model calibration variables. Conceptually, we can write this as:

$$estimate = project * calibration$$

The estimate variance is hence a function of *variance in the project options* and the *space of possible calibrations*. Traditional approaches use historical data to reduce the space of possible calibrations (e.g. using regression). In our approach, we leave the calibration variables unconstrained and instead use an AI search engine to reduce the space of possibilities in the project options. In numerous studies (including one reported last

Reuse Iteration	Description
First reuse	Using software from a previous project for the first time.
Second reuse	Reusing software from a previous project for the second time
Third reuse	Reusing the same software in a new project for the third time.
Fourth reuse	Reusing software using a mature reuse approach, tools, and personnel.

Fig. 1. Process changes imposed by implementing reuse incrementally.

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.

Fig. 2. Nine drastic changes from [9].

year at ICSP'08 and elsewhere [6–8]) we showed that this methods can yield estimates close to those seem using traditional methods, without requiring a time consuming data collection exercise.

In this paper, we use STAR to comparatively assess 13 possible changes to a project. Figure 1 shows four changes to a project based on reuse while Figure 2 defines some alternatives. These alternatives are drastic changes a project manager could implement in an effort to reduce effort, schedule and defects in a particular project [9]. Our results will show that some projects gain the most benefit from applying reuse, while there are often other changes (such as those listed in Figure 2) that can be more effective. Hence, we recommend assessing the value of reuse on a project-by-project basis. Process simulation tools are useful for making such an assessment and tools like STAR are especially useful when there is insufficient data for local calibration.

In the remainder of this paper, we present background information about software reuse and process estimation models. Then we document the simulation approach utilized to evaluate the effects of adopting software reuse compared to alternative strategies for two NASA case studies.

2 The Models: COCOMO and COQUALMO

For this study we use two USC software process models. The COQUALMO software *defect* predictor [10, p254-268] models two processes (defect introduction and defect removal) for three phases (requirements, design, and coding). Also, the COCOMO software *effort* and development *time* predictor [10, p29-57] estimates development months (225 hours) and calendar months and includes all coding, debugging, and management activities. COCOMO assumes that effort is exponentially proportional to some *scale factors* and linearly proportional to some *effort multipliers*.

	Definition	Low-end = {1,2}	Medium ={3,4}	High-end= {5,6}
Defect removal features				
execution-based testing and tools (etat)	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 (pr)	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 risks 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 35%	35% - 90%	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

Fig. 3. Features of the COCOMO and COQUALMO models used in this study.

From our perspective, these models have several useful features. Unlike other models such as PRICE-S [11], SLIM [12], or SEER-SEM [13], the COCOMO family of models is fully described in the literature. Also, at least for the effort model, there exist baseline results [14]. Also, we work extensively with government agencies writing software. Amongst those agencies, these models are frequently used to generate and justify budgets. Further, The space of possible tunings within COCOMO & COQUALMO is well defined. Hence, it is possible to explore the space of possible tunings.

The process simulation community (e.g., Raffo [15]) studies models far more elaborate than COCOMO or COQUALMO. For example, COCOMO & COQUALMO assume linear parametric equations while other researchers explore other forms:

- discrete-event models [16];
- system dynamics models [17];
- state-based models [18];
- rule-based programs [19];
- standard programming constructs such as those used in Little-JIL [20].

These rich modeling frameworks allow the representation of detailed insights into an organization. However, the effort required to tune them is non-trivial. For example, Raffo spent two years tuning and validating one of such models to one particular site [5]. Also, we have found that the estimation variance of COCOMO can be reduced via intelligent selection of input variables, even allowing for full variance in the tuning parameters. We would consider switching to other models if it could be shown that the variance of these other models could be controlled just as easily.

Our models use the features presented in Figure 3. This figure lists a variety of project *features* with the *range* {very low, low, nominal, high, very high, extremely high} or $\{vl = 1, l = 2, n = 3, h = 4, vh = 5, xh = 6\}$. For specific projects, not all features are known with certainty. For example, until software is completed, the exact size of a program may be unknown. Hence, exploring our effort, schedule, and defect models requires exploring a large trade-space of possible model inputs.

2.1 Effect of Reuse on Model Parameters

The effects of ad-hoc software reuse can be mapped to changes to settings to the COCOMO parameters. For instance, programmers capability (pcap) inherently increases every time a piece of software is reused given that in the process the same programmer is employed. This is the case with NASA spacecraft software, where reuse can be found within the same software development company and where the software modules are signed by the same developers [21]. Similarly, we can assume direct inherent effects to the analyst capability (acap), the application experience (apex), the analyst capability (acap), the precedence of the software (prec), the process maturity (pmat), and the language and tool experience (ltex). On the other hand, the software platform must remain fairly unchanged throughout reuses so software pieces can be reused with ease. For this reason we assume that platform volatility has to decrease as a piece of software is reused from project to project.

A final assumption on the size of the software comes from the observation that the progressive reuse of software components allows the construction of more sizeable systems. In our simulations we assume that the code base increases from system to system by 25%.

Incremental Reuse	Effects on Projects

1 First Reuse	acap = ACAP$_L$; apex = APEX$_L$; pcap = PCAP$_L$; prec = PREC$_L$
	pmat = PMAT$_L$; ltex = LTEX$_L$; pvol = PVOL$_H$; kloc = KLOC$_L$

2 Second Reuse	acap = ACAP$_L$+1 ; apex = APEX$_L$+1 ; pcap = PCAP$_L$+1 ; prec = PREC$_L$+1
	pmat = PMAT$_L$+1 ; ltex = LTEX$_L$+1 ; pvol = PVOL$_H$-1 ; kloc = KLOC$_L$*1.25

3 Third Reuse	acap = ACAP$_L$+2 ; apex = APEX$_L$+2 ; pcap = PCAP$_L$+2 ; prec = PREC$_L$+2
	pmat = PMAT$_L$+2 ; ltex = LTEX$_L$+2 ; pvol = PVOL$_H$-2 ; kloc = KLOC$_L$*1.5625

4 Fourth Reuse	acap = ACAP$_L$+3 ; apex = APEX$_L$+3 ; pcap = PCAP$_L$+3 ; prec = PREC$_L$+3
	pmat = PMAT$_L$+3 ; ltex = LTEX$_L$+3 ; pvol = PVOL$_H$-3 ; kloc = KLOC$_L$*1.953125

Fig. 4. Implementing software reuse incrementally.

Figure 4 shows the constraints we claim software reuse imposes on model parameters for a given project. In the figure the variable X_L represents the lowest value in the range of the model variable X. Similarly, X_H represents the highest value for the X variable. In order to impose the constraint of reuse in a particular project we increase the X_L and lower the X_H for the particular variable X according to the logic above. For instance, let's say that for a particular project P, pcap ranges between 2 and 4, and pvol ranges between 3 and 5. In this case, PCAP$_L = 2$, PCAP$_H = 4$, PVOL$_L = 3$, and PVOL$_H = 5$. if we imposed a "Second Reuse" strategy, pcap would be set to 3 (PCAP$_L + 1$), and pvol would be set to 4 (PVOL$_H - 1$).

2.2 Defining the Alternatives to Reuse

Similarly to the constraints imposed by the incremental software reuse strategies, Figure 5 defines the values we imposed on each case study as part of each drastic change. Most of the values in Figure 5 are self-explanatory with two exceptions. Firstly, the $kloc * 0.5$ in "reduce functionality" means that, when imposing this drastic change, we only implement half the system. Secondly, most of the features fall in the range one to five. However, some have minimum values of 2 or higher (e.g., $pvol$ in "improve tools/tech/dev"), and some have maximum values of 6 (e.g., $site$ in "improve tools/tech/dev"). This explains why some of the drastic changes result in values other than one or five.

To impose a drastic change on a case study, if that change refers to feature X (in the right-hand column of Figure 5), then we first (a) removed X from the values and ranges of the case study (if it was present); then (b) added the changes of Figure 5 as fixed values for that case study.

3 Case Studies

Recall that the goal of our study is to analyze simulations of process estimation using COCOMO and COQUALMO models on typical NASA projects. Our purpose is to evaluate the relative merits of adopting software reuse compared to other project improvement strategies. The comparison is based on effort, quality, and schedule measured in person-months, defects per KSLOC, and months, respectively. In this study we explore the perspective of the Project Manager in the context of NASA software development.

Drastic change	Effects on Projects

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

Fig. 5. Implementing drastic changes.

	ranges			values	
project	feature	low	high	feature	setting
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
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		

Fig. 6. Two case studies. Numeric values $\{1, 2, 3, 4, 5, 6\}$ map to $\{verylow, low, nominal, high, veryhigh, extrahigh\}$.

Figure 6 partially describes the two NASA case studies we explore in terms of the COCOMO and COQUALMO input parameters. Both case studies reflect typical ranges seen at NASA's Jet Propulsion Laboratory [7].

Inside our model, project choices typically range from 1 to 5 where "3" is the nominal value that offers no change to the default estimate. Some of the project choices in Figure 6 are known precisely (see all the choices with single *values*). But many of the features in Figure 6 do not have precise values (see all the features that *range* from some *low* to *high* value).

We evaluate the effects of the project improvement strategies on the case studies above using STAR: a Monte Carlo engine augmented by a simulated annealer.

STAR runs as follows. First, a project P is specified as a set of min/max ranges to the input variables of STAR's models:

– If a variable is known to be exactly x, then then $min = max = x$.
– Else, if a variable's exact value is not known but the range of possible values is known, then min/max is set to the smallest and largest value in that range of possibilities.
– Else, if a variable's value is completely unknown then min/min is set to the full range of that variable in Figure 3.

Second, STAR's simulated annealer[3] seeks constraints on P that most improve the model's score. A particular subset of $P' \subseteq P$ is scored by using P' as inputs to the COCOMO and COQUALMO. When those models run, variables are selected at random from the min/max range of possible tunings T and project options P. In practice, the majority of the variables in P' can be removed without effecting the score; i.e. our models exhibit a *keys effect* where a small number of variables control the rest [23]. Finding that minimal set of variables is very useful for management since it reveals the *least* they need to change in order to *most* improve the outcome. Hence, simulated annealing, STAR takes a third step.

In this third step, a Bayesian sensitivity analysis finds the smallest subset of P' that most effects the output. The scores seen during simulated annealing are sorted into the (10,90)% (best,rest) results. Members of P' are then ranked by their Bayesian probability of appearing in $best$. For example, $10,000$ runs of the simulated annealer can be divided into 1,000 lowest *best* solutions and 9,000 *rest*. If the range $rely = vh$ might appears 10 times in the $best$ solutions, but only 5 times in the $rest$ then:

$$\begin{aligned}
E &= (reply = vh)\\
P(best) &= 1000/10000 = 0.1\\
P(rest) &= 9000/10000 = 0.9\\
freq(E|best) &= 10/1000 = 0.01\\
freq(E|rest) &= 5/9000 = 0.00056\\
like(best|E) &= freq(E|best) \cdot P(best) = 0.001\\
like(rest|E) &= freq(E|rest) \cdot P(rest) = 0.000504\\
P(best|E) &= \frac{like(best|E)}{like(best|E) + like(rest|E)} = 0.66
\end{aligned} \tag{1}$$

Equation 1 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 Equation 1 with a support term. Support should $increase$ as the frequency of a range $increases$, i.e. $like(x|best)$ is a valid support measure. STAR1 hence ranks ranges via

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

After ranking members of P', STAR then imposes the top i-th ranked items of P' on the model inputs, then running the models 100 times. This continues until the scores seen using $i + 1$ items is not statistically different to those seen using i (t-tests, 95% confidence). STAR returns items $1..i$ of P' as the *least* set of project decisions that *most* reduce effort, defects, and development time. We call these returned items the *policy*.

[3] Simulated annealers randomly alter part of the some $current$ solution. If this new solution scores better than the current solution, then $current = new$. Else, at some probability determined by a temperature variable, the simulated annealer may jump to a sub-optimal new solution. Initially the temperature is "hot" so the annealer jumps all over the solution space. Later, the temperature "cools" and the annealer reverts to a simple hill climbing search that only jumps to new better solutions. For more details, see [22].

Note that STAR constraints the project options P but never the tuning options T. That is, the *policy* generated by STAR contains parts of the project options P that most improve the score, despite variations in the tunings T. This approach has the advantage that it can reuse COCOMO models without requiring local tuning data.

Previously [7] we have shown that this approach, that does not use local tuning, generates estimates very similar to those generated after using local tuning via the "LC" method proposed by Boehm and in widespread use in the COCOMO community [24]. We have explained this effect as follows. Uncertainty in the project options P and the tuning options T contribute to uncertainty in the estimates generated by STAR's models. However, at least for the COCOMO and COQUALMO models used by STAR, the uncertainty created by P dominates that of T. Hence, any uncertainty in the output can be tamed by constraining P and not T.

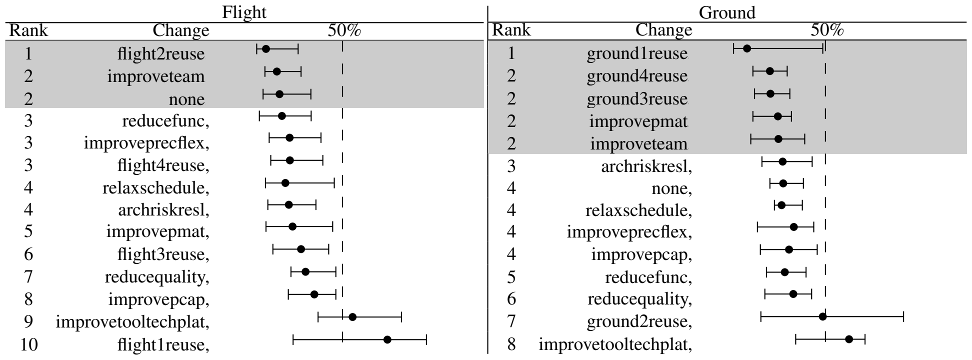
The reader may wonder why we use an stochastic method like STAR: would not a simpler method suffice? For example, Many of the relationships inside COCOMO model are linear and a simple linear extrapolation across the space of possibilities could assess the relative effectiveness of different changes. In reply, we note that:

- Even after tuning the gradient of the relationships may not be known with certainty. For example, in the COCOMO effort model, predictions are affected linearly and exponentially by two types of input parameters; the new project data and the historical dataset. In COCOMO this results in the coefficients, a and b, which define the relationship between size and effort. Baker [25] tuned these a, b values using data from NASA systems. After thirty 90% random samples of that data, the a, b ranges were surprisingly large: $(2.2 \leq a \leq 9.18) \wedge (0.88 \leq b \leq 1.09)$. Baker's results forced a rethinking of much our prior work in this area. Instead of exploring better learners for local calibration, now we use tools like STAR to search models for conclusions that persist across the space of possible calibrations.
- Simplistic linear extrapolation may be inappropriate when optimizing for effort *and* time *and* defects, there may be contradictory effects. For example, we have results where reducing effort leads to a dramatic increase in defects [8]. 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 (e.g. using a tool that considers combinations of effects, like STAR).

4 Results

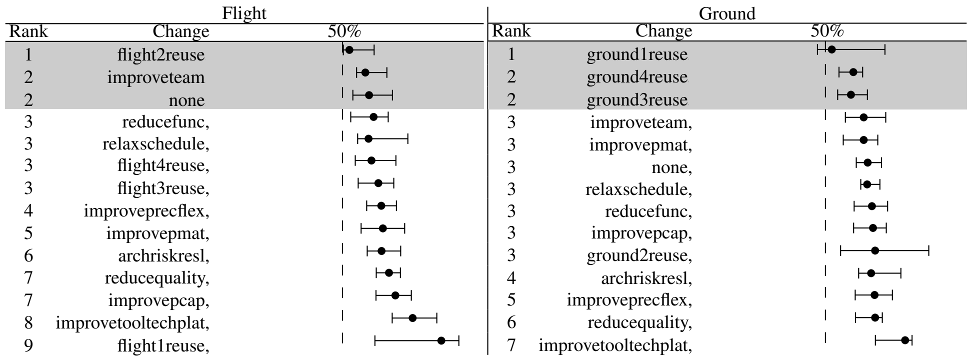
For each case study of Figure 6, STAR searches within the ranges to find constraints that most reduce development effort, development time, and defects. The results are shown in Figure 7, Figure 8, and Figure 9. In those figures:

- All results are normalized to run 0..100, min..max.
- Each row shows the 25% to 75% quartile range of the normalized scores collected during the simulation.
- The median result is shown as a black dot.
- All the performance scores get *better* when the observed scores get *smaller*.



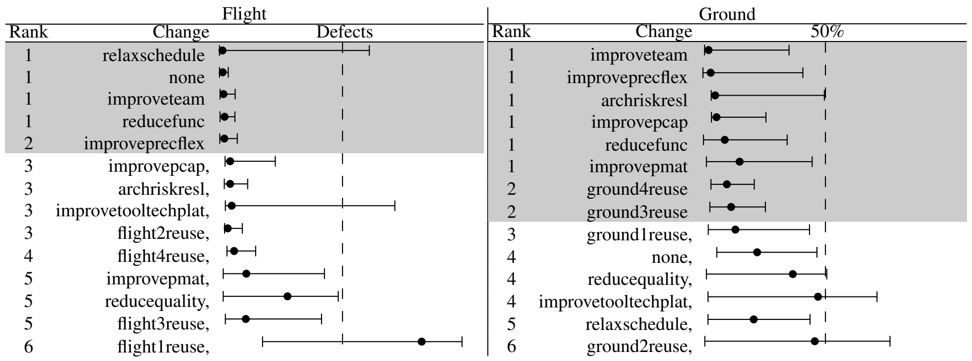
Flight Rank	Flight Change	Ground Rank	Ground Change
1	flight2reuse	1	ground1reuse
2	improveteam	2	ground4reuse
2	none	2	ground3reuse
3	reducefunc,	2	improvepmat
3	improveprecflex,	2	improveteam
3	flight4reuse,	3	archriskresl,
4	relaxschedule,	4	none,
4	archriskresl,	4	relaxschedule,
5	improvepmat,	4	improveprecflex,
6	flight3reuse,	4	improvepcap,
7	reducequality,	5	reducefunc,
8	improvepcap,	6	reducequality,
9	improvetooltechplat,	7	ground2reuse,
10	flight1reuse,	8	improvetooltechplat,

Fig. 7. EFFORT: staff months (normalized 0..100%): top-ranked changes are shaded.



Flight Rank	Flight Change	Ground Rank	Ground Change
1	flight2reuse	1	ground1reuse
2	improveteam	2	ground4reuse
2	none	2	ground3reuse
3	reducefunc,	3	improveteam,
3	relaxschedule,	3	improvepmat,
3	flight4reuse,	3	none,
3	flight3reuse,	3	relaxschedule,
4	improveprecflex,	3	reducefunc,
5	improvepmat,	3	improvepcap,
6	archriskresl,	3	ground2reuse,
7	reducequality,	4	archriskresl,
7	improvepcap,	5	improveprecflex,
8	improvetooltechplat,	6	reducequality,
9	flight1reuse,	7	improvetooltechplat,

Fig. 8. MONTHS: calendar (normalized 0..100%): top-ranked changes are shaded.



Flight Rank	Flight Change	Ground Rank	Ground Change
1	relaxschedule	1	improveteam
1	none	1	improveprecflex
1	improveteam	1	archriskresl
1	reducefunc	1	improvepcap
2	improveprecflex	1	reducefunc
3	improvepcap,	1	improvepmat
3	archriskresl,	2	ground4reuse
3	improvetooltechplat,	2	ground3reuse
3	flight2reuse,	3	ground1reuse,
4	flight4reuse,	4	none,
5	improvepmat,	4	reducequality,
5	reducequality,	4	improvetooltechplat,
5	flight3reuse,	5	relaxschedule,
6	flight1reuse,	6	ground2reuse,

Fig. 9. Defect / KLOC (normalized 0..100%): top-ranked changes are shaded.

– The "none" row comes from Monte Carlo simulations of the current ranges, without any changes.

In each figure, the rows are sorted by the number of times a change *looses* to other changes. In order to assess number of *losses*, we used the Mann-Whitney test at 95% confidence test (this test was chosen since (a) due to the random nature of Monte Carlo simulations, the inputs to each run are not paired; and (b) ranked tests make no, possibly inappropriate, assumption about normality of the results). Two rows have the same rank if there is no statistical difference in their distributions. Otherwise, we compare the number of losses.

The shaded rows in Figures 7, 8, and 9 mark the top ranked changes. Observe how, with Ground systems, first and second generation reuse always appears in the top rank for effort, months, and defects. That is, for these systems, two generations of reuse is as good, or better, than other proposed changes to a project. That is, for NASA ground software a case can be made in support of software reuse as it is shown to be as good or better than the other strategies.

The results are quite different for Flight systems where all reuse methods are absent from the top ranked changes. In fact, no reuse change appears in the Flight results till rank three or above and all reuse strategies rank below ”none” (meaning that adopting no strategy could yield better results). For NASA flight software, no case can be made for adopting software reuse.

What is surprising is how different the strategies work for similar projects such as the ones presented in this study. The only difference between NASA ground and flight systems lies in size (KLOC), reliability (rely), and complexity(cplx). Ground tends to have lower values for these ranges. Other than that, both case studies have the same ranges and values.

Another finding worth mentioning is how ”improve team” consistently ranks top along side with reuse for ground systems. This might be related to the notion that ”sociology beats technology in terms of successfully completing projects,” [26] or it might be at least comparable.

5 External Validity

Our results make no use of local calibration data. That is, our results are not biased by the historical record of different sites. Hence, in this respect, our results score very highly in terms of external validity.

However, in another respect, the external validity of our results is highly biased by the choice of underlying model (COCOMO, COQUALMO); and the range of changes and projects we have explored:

– *Biases from the model:* While the above results hold over the entire space of calibration possibilities of COCOMO/COQUALMO, then may *not* hold for different models. One reason we are staying with COCOMO/COQUALMO (at least for the time being) is that we have shown that STAR can control these models without requiring local calibration data. We find this to be a compelling reason to prefer these models.
– *Biased from the range of cases explored:* Another threat to the external validity of our models is the range of changes explored in this study. This paper has only

ranked reuse against the changes listed in Figure 2. Our changes may do better than and we will explore those in future work.

– *Biases from our selected case studies:* Lastly, we have only explored the projects listed in Figure 6. We are currently working towards a more extensive study where we explore more projects, including projects that do not come from NASA.

6 Conclusion

Our reading of the literature is that much prior assessment of reuse has focused on a very narrow range of of issues. Here, we have tried broadening the debate by assessing reuse with respect to the broader context of minimizing effort and defects and development time.

This paper has explored the case of (a) ranking reuse against different effort, schedule, and defect reduction strategies using (b) models with competing influences that (c) have not been precisely tuned using local data. In this case, we avoided the need for local data to calibrate the models via using the STAR tool. STAR leaves the calibration variables of a model unconstrained, then uses AI search to find project options that most reduces effort, development time, and defects.

STAR was applied here to a study of four incremental reuse strategies and the eight drastic changes. These 13 project changes were applied to two NASA case studies. We found that reuse strategies in general performed as well or better than drastic change strategies on ground software, but did worse than adopting no strategy in the case of flight software systems.

These results suggest that project managers looking for implementing software reuse into their projects may find worthwhile checking the relative merits of reuse against other project improvement options. That is, the relative merits of software reuse should be evaluated in a project-by-project basis. reuse strategies against other project improvement strategies.

In conclusion, in theory, software reuse is an attractive approach to any software project capable of adopting it. However, in practice, reuse might not be the most useful strategy and changing something else (or changing nothing at all) could be more beneficial. AI search over process simulation models is useful for finding the best changes, particularly when there is not sufficient data for precisely tuning a simulation model.

References

1. Trauter, R.: Design-related reuse problems an experience report. In: Proceedings of the International Conference on Software Reuse. (1998)
2. Poulin, J.S.: Measuring Software Reuse. Addison Wesley (1997)
3. Lions, J.: Ariane 5 flight 501 failure (July 1996) Available from: `http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html`.
4. Leveson, N.G., Turner, C.S.: An investigation of the therac-25 accidents. IEEE Computer **26**(7) (July 1993) 18–41
5. Raffo, D.: Modeling software processes quantitatively and assessing the impact of potential process changes of process performance (May 1996) Ph.D. thesis, Manufacturing and Operations Systems.

6. Menzies, T., Elwaras, O., Hihn, J., Feathear, M., Boehm, B., Madachy, R.: The business case for automated software engineerng. In: IEEE ASE. (2007) Available from `http://menzies.us/pdf/07casease-v0.pdf`.
7. Menzies, T., Elrawas, O., Barry, B., Madachy, R., Hihn, J., Baker, D., Lum, K.: Accurate estiamtes without calibration. In: International Conference on Software Process. (2008) Available from `http://menzies.us/pdf/08icsp.pdf`.
8. Menzies, T., Williams, S., El-waras, O., Boehm, B., Hihn, J.: How to avoid drastic software process change (using stochastic statbility). In: ICSE'09. (2009)
9. Boehm, B., In, H.: Conflict analysis and negotiation aids for cost-quality requirements. Software Quality Professional **1**(2) (March 1999) 38–50
10. Boehm, B., Horowitz, E., Madachy, R., Reifer, D., Clark, B.K., Steece, B., Brown, A.W., Chulani, S., Abts, C.: Software Cost Estimation with Cocomo II. Prentice Hall (2000)
11. Park, R.: The central equations of the price software cost model. In: 4th COCOMO Users Group Meeting. (November 1988)
12. Putnam, L., Myers, W.: Measures for Excellence. Yourdon Press Computing Series (1992)
13. Jensen, R.: An improved macrolevel software development resource estimation model. In: 5th ISPA Conference. (April 1983) 88–92
14. Chulani, S., Boehm, B., Steece, B.: Bayesian analysis of empirical software engineering cost models. IEEE Transaction on Software Engineerining **25**(4) (July/August 1999)
15. Raffo, D., Menzies, T.: Evaluating the impact of a new technology using simulation: The case for mining software repositories. In: Proceedings of the 6th International Workshop on Software Process Simulation Modeling (ProSim'05). (2005)
16. Kelton, D., Sadowski, R., Sadowski, D.: Simulation with Arena, second edition. McGraw-Hill (2002)
17. Abdel-Hamid, T., Madnick, S.: Software Project Dynamics: An Integrated Approach. Prentice-Hall Software Series (1991)
18. Martin, R., Raffo, D.M.: A model of the software development process using both continuous and discrete models. International Journal of Software Process Improvement and Practice (June/July 2000)
19. Mi, P., Scacchi, W.: A knowledge-based environment for modeling and simulation software engineering processes. IEEE Transactions on Knowledge and Data Engineering (September 1990) 283–294
20. Cass, A., Lerner, B.S., McCall, E., Osterweil, L., Jr., S.M.S., Wise, A.: Little-jil/juliette: A process definition language and interpreter. In: Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000). (June 2000) 754–757
21. Orrego, A.: Software reuse study report. Technical report, NASA IV&V Facility, Fairmont, WV, USA. (April 2005)
22. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science, Number 4598, 13 May 1983 **220, 4598** (1983) 671–680
23. Menzies, T., D.Owen, Richardson, J.: The strangest thing about software. IEEE Computer (2007) `http://menzies.us/pdf/07strange.pdf`.
24. Boehm, B.: Software Engineering Economics. Prentice Hall (1981)
25. Baker, D.: 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`.
26. DeMarco, T., Lister, T.: Peopleware: productive projects and teams. Dorset House Publishing Co., Inc., New York, NY, USA (1987)