



On the Relative Merits of Software Reuse



West Virginia University.

Andres Orrego [WVU]

Tim Menzies [WVU]

Ous El-Rawas [WVU]

andres_orrego@yahoo.com

tim@menzies.us

oeIrawas@mix.wvu.edu



Outline

- ➔ Introduction
 - SW reuse
- ➔ Problem Statement
 - When does reuse work?
- ➔ Research Planning
 - How to assess project improvement strategies
- ➔ Operation
 - Reuse vs. alternatives
- ➔ Case study
 - Data analysis
 - Interpretation of Results
- ➔ Conclusions
- ➔ Future Work
- ➔ Questions?



Reuse Definitions

- ⇒ Reuse: Using a software asset in the solution of a different problem [*IEEE Standard 1517-1999*]
 - Ad hoc: Unplanned (“Accidental”)
 - Systematic: Planned (from Reusable S/W Components)
 - Black-box: Unmodified (“Plug ‘n Play”)
 - White-box: Modified (“Reengineering”)
- ⇒ Ad-hoc, white-box reuse is most common



Software Reuse

Benefits

- ➔ Lower costs
- ➔ Increased productivity
- ➔ Easier maintenance
- ➔ Higher quality
- ➔ Lower risk
- ➔ Shorter lifecycle
- ➔ Better interoperability

Issues

- ➔ Higher initial cost
- ➔ Lack of standard metrics
- ➔ Not always the best choice
- ➔ Hard to implement
- ➔ Benefits are hard to quantify



Reuse in Industry

Org.	Results	Approach	Refs.
BTG, Inc.	↓ TTM (Time to Market)	Accidental, DA (Domain Analysis)	[Incorvaia90]
Raytheon	↑ R/U (Reuse)	Library, tools, training	[Incorvaia90]
NTT	↑ R/U, ↑ PDTY (Productivity)	R/U and tool teams, R/U incentives, cataloging RSC	[Isoda91]
Motorola	↑ PDTY, ↑ QTY (Quality)	R/U team, Mgmt support, training, R/U incentives, tools	[Joos94]
HP	↑ QTY, ↓ TTM, ↑ PDTY, ↑ ROI	Systematic Black-box, libraries, Reengineering to RSC, tools	[Lim94]
Europe	9/24 (or 37.5%) reuse attempts failed	Mgmt support, repository, DA, R/U team, R/U incentives, training	[Morisio02]



Hypotheses

⇒ H1:

- Adopting software reuse is better at reducing effort, schedule, and defects than doing nothing

⇒ H2:

- Adopting software reuse is better at reducing effort, schedule, and defects than alternative project changes



Boehm99: If not reuse, then...

<i>Improve process maturity</i>	Implementing standardized processes
<i>Improve personnel</i>	Firing and rehiring employees
<i>Improve development environment</i>	Changing OS, IDEs, languages
<i>Improve precedentness / flexibility</i>	Changing project goals and methods
<i>Increase archit. analysis / risk resolutn.</i>	Far more elaborate life cycle analysis
<i>Improve the team</i>	Implementing team building efforts
<i>Relax schedule</i>	Delivering the system later
<i>Reduce functionality</i>	Delivering less than expected
<i>Reduce quality</i>	Skipping testing, code reviews, etc.



Models Used

- ➔ COCOMO effort and time estimation model
 - 5 scale factors
 - 17 effort multipliers
- ➔ COQUALMO defect estimation model
 - COCOMO input parameters plus:
 - Execution based testing tools
 - Automated analysis
 - Peer reviews



Dodging the Calibration Problem

- ⇒ Estimate = projectDetails * modelCalibration
 - Estimate error = projectError and calibrationError

- ⇒ We must have accurate modelCalibration when...

$$\text{Estimate} = \text{projectDetails} * \text{modelCalibration}$$

- ⇒ But we don't when...

$$\text{Estimate} = \text{projectDetails} * \text{modelCalibration}$$

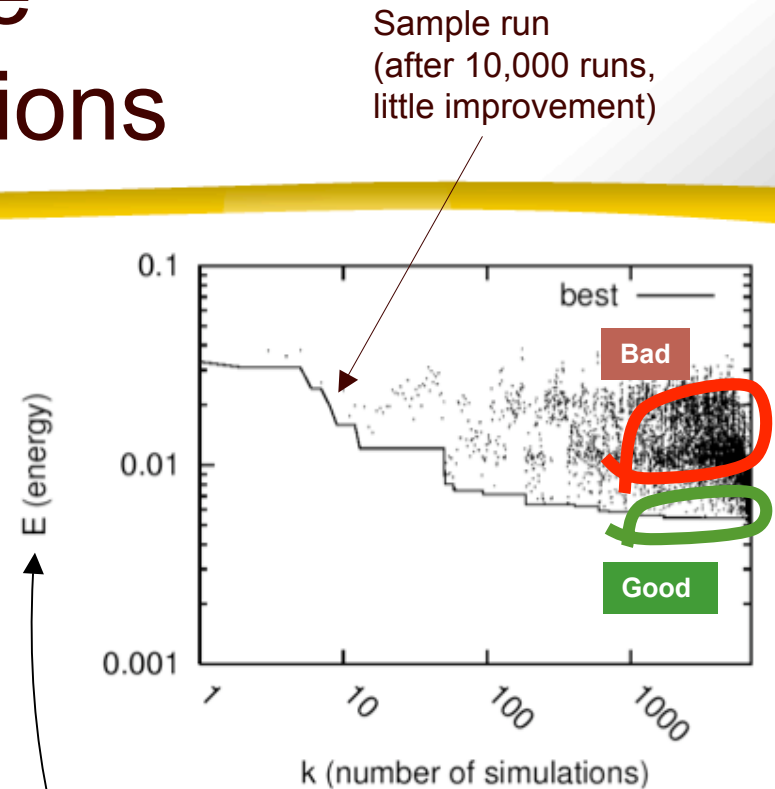


STAR: A Strategy Assessment

- ➔ Monte Carlo sampling over ...
 - ... the project options left after imposed constraints
- ➔ Apply AI search methods to select
 - Project options that most improve the estimate

Searching the space of options + calibrations


- Using simulated annealing, Monte Carlo simulated annealing across intersection of
 - A particular project type
 - Space of possible tunings
- Rank options by frequency in **good**, not **bad**
- For r options
 - Try setting the $1 \leq x \leq R$ top ranked options
 - Simulate (100 times) to check the effect of options 1 .. x
- Smile if
 - Reduced median and variance in defects/ efforts/ time/ threats



But what is the
Performance score?

Automatically sampling
across space of possibilities

Note: no
calibration



What is the space of project options?

- ⇒ The typical project input parameters seen at NASA's Jet Propulsion Laboratory for:
- Flight Systems
 - Ground systems

“Values” = fixed

“Ranges” = Loose (select within these ranges)

project	ranges			values	
	feature	low	high	feature	setting
Flight:	rely	3	5	tool	2
	data	2	3	seed	3
	cplx	3	6		
	time	3	4		
	stor	3	4		
	pvol	2	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	Ksloc	7	418		
Ground:	rely	1	4	tool	2
	data	2	3	seed	3
	cplx	1	4		
	time	3	4		
	stor	3	4		
	pvol	2	4		
	acap	3	5		
	apex	3	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	Ksloc	11	392		



Incremental Reuse Effects

- ⇒ Reusing software over time results in
 - increased
 - Analyst's capability
 - Programmer's capability
 - Application experience
 - Software precedentness
 - Process maturity
 - Language and toolset experience
 - Reduced
 - Platform volatility



Reuse Effects on Model Inputs

Reuse iteration	Project parameter constraints
First Reuse	$acap = ACAP_L$; $apex = APEX_L$; $pcap = PCAP_L$; $prec = PREC_L$ $pmat = PMAT_L$; $ltex = LTEX_L$; $pvol = PVOL_H$; $kloc = KLOC_L$
Second Reuse	$acap = ACAP_{L+1}$; $apex = APEX_{L+1}$; $pcap = PCAP_{L+1}$; $prec = PREC_{L+1}$ $pmat = PMAT_{L+1}$; $ltex = LTEX_{L+1}$; $pvol = PVOL_H-1$; $kloc = KLOC_L * 1.25$
Third Reuse	$acap = ACAP_{L+2}$; $apex = APEX_{L+2}$; $pcap = PCAP_{L+2}$; $prec = PREC_{L+2}$ $pmat = PMAT_{L+2}$; $ltex = LTEX_{L+2}$; $pvol = PVOL_H-2$; $kloc = KLOC_L * (1.25)^2$
Fourth Reuse	$acap = ACAP_{L+3}$; $apex = APEX_{L+3}$; $pcap = PCAP_{L+3}$; $prec = PREC_{L+3}$ $pmat = PMAT_{L+3}$; $ltex = LTEX_{L+3}$; $pvol = PVOL_H-3$; $kloc = KLOC_L * (1.25)^3$



Effects of Other Changes

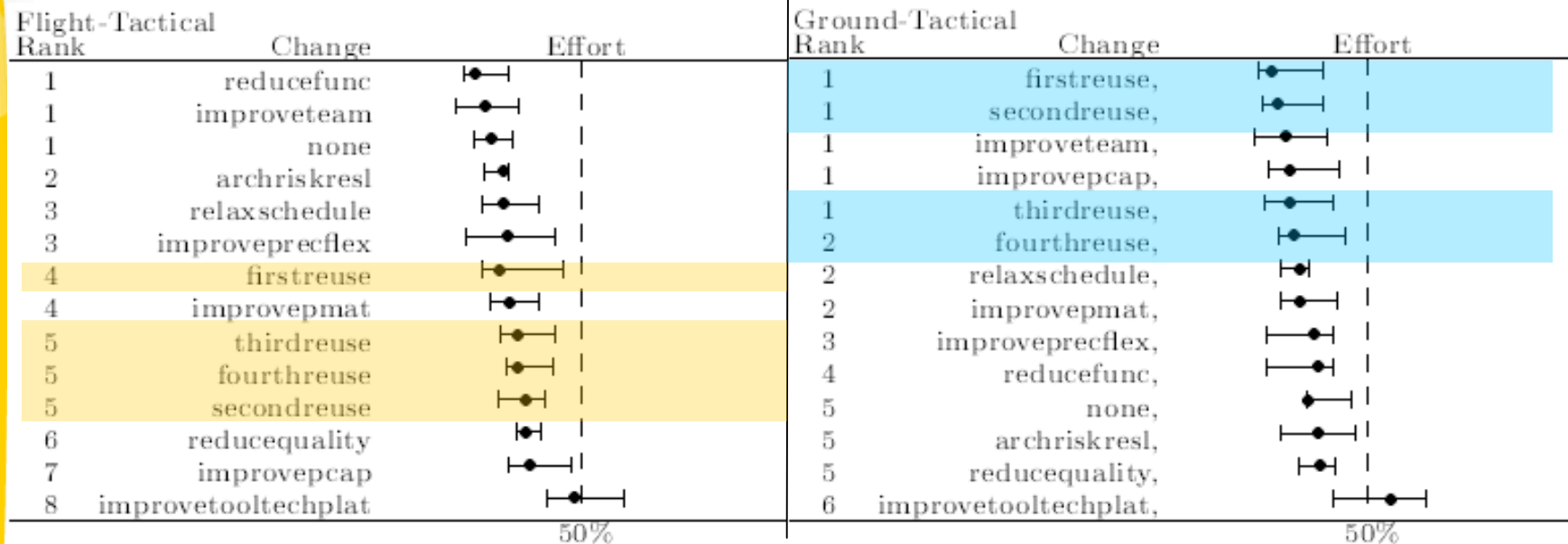
Drastic change	Project parameter constraints
Improve personnel	acap=pcap=pcon=apex=plex=ltex=5
Improve development environment	time=stor=3; pvol=2; tool=5; site=6
Improve precedentness / flexibility	prec=flex=5
Increase archit. analysis / risk resolution	resl=5
Relax schedule	sced=5
Improve process maturity	pmat=5
Reduce functionality	data=2;kloc*0.5
Improve the team	team=5
Reduce quality	rely=docu=cplx=1; time=3



Data Analysis

- ➔ Run STAR 100 times per strategy (incremental reuse + alternative changes) for both projects.
- ➔ Observations are compared to each other based on non-parametric, Mann-Whitney tests @ 95% confidence.
- ➔ Tallies of win, losses, and ties were kept
- ➔ Treatment ranked according to number of losses in ascending order
- ➔ Top strategies are better for reducing estimates

The Results: On Effort



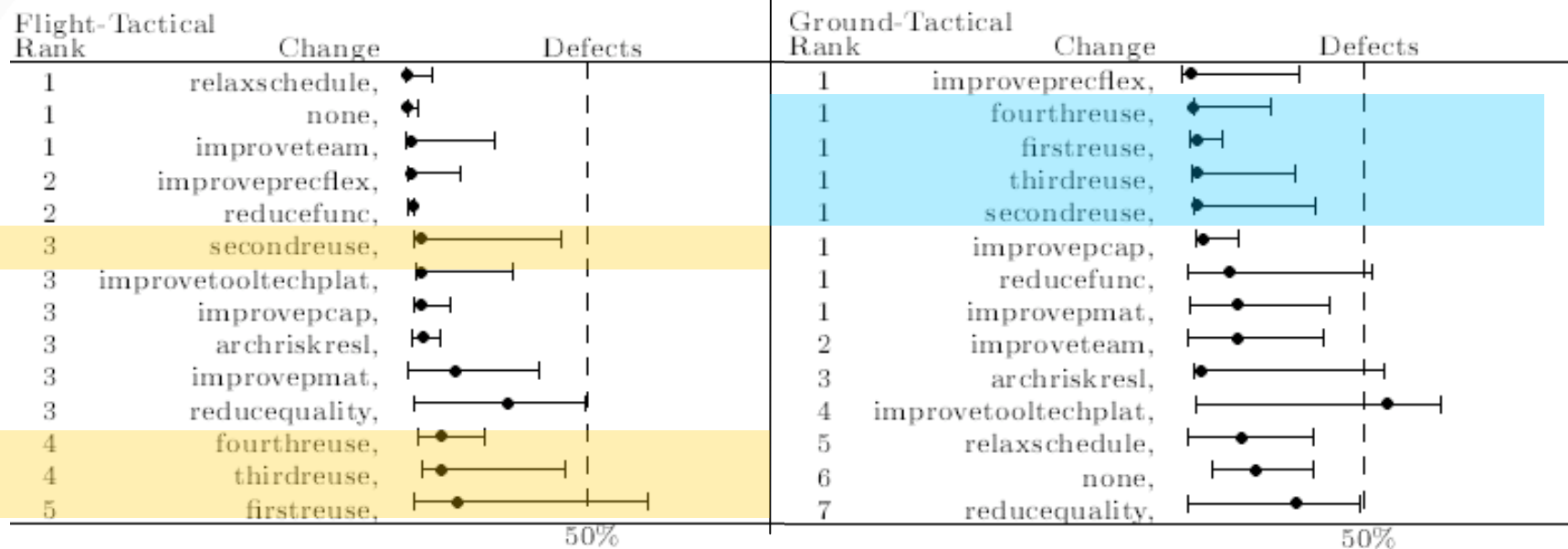
EFFORT = total staff months (normalized 0..100%)

The Results: On Schedule



MONTHS = calendar months (normalized 0..100%)

The Results: On Defects



DEFECTS = Defects/KLOC (normalized 0..100%)



Interpretation of Results

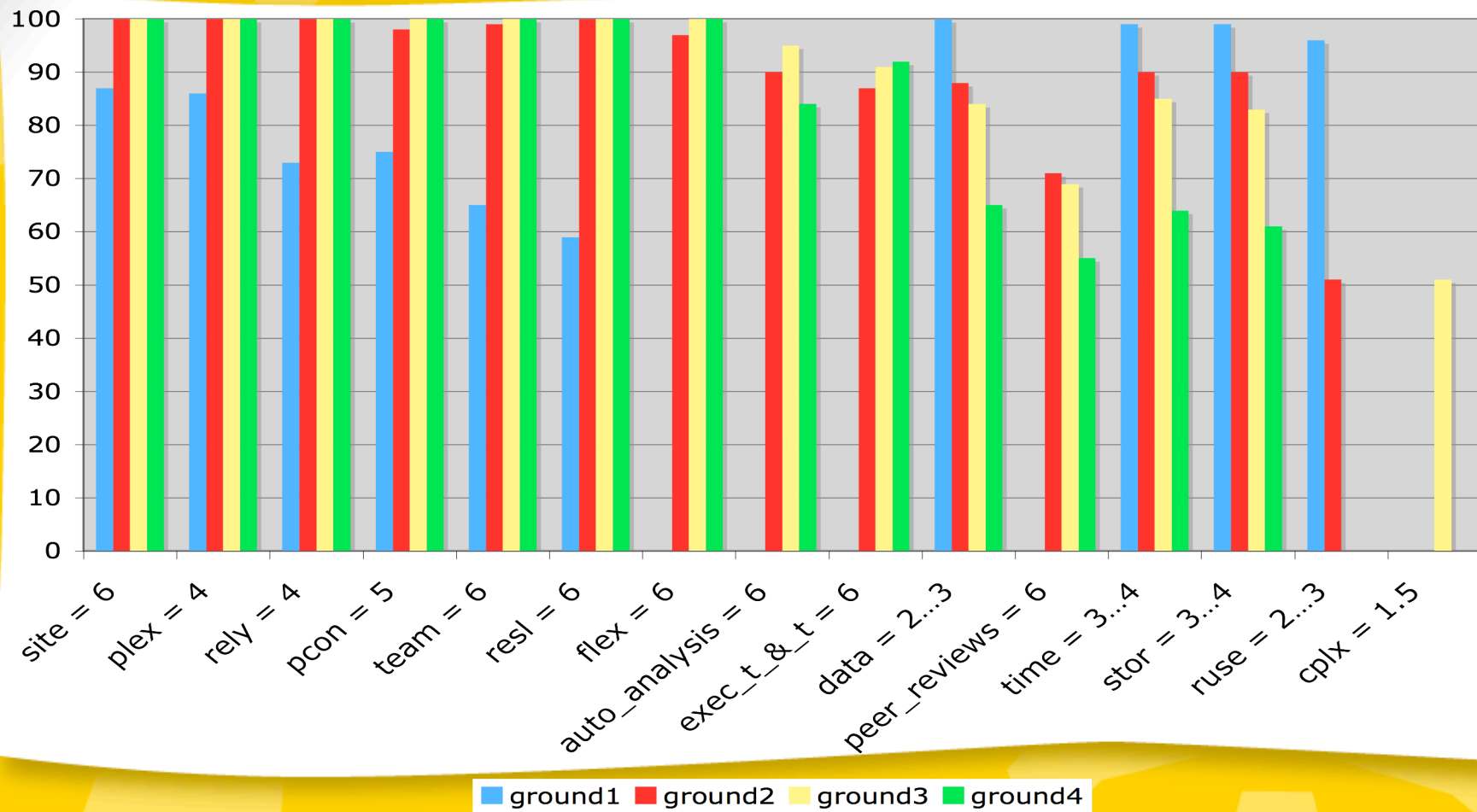
- ⇒ Reuse is better or as good as any other strategy on NASA ground systems (cannot reject H2 for ground systems)
- ⇒ No case for reuse can be found on NASA flight systems. (Reject H1 for flight systems: “Doing nothing” is a better treatment)
- ⇒ When optimizing for effort and defects any reuse strategy has merits for ground software
- ⇒ When optimizing for months, a case can be made for stopping at second reuse



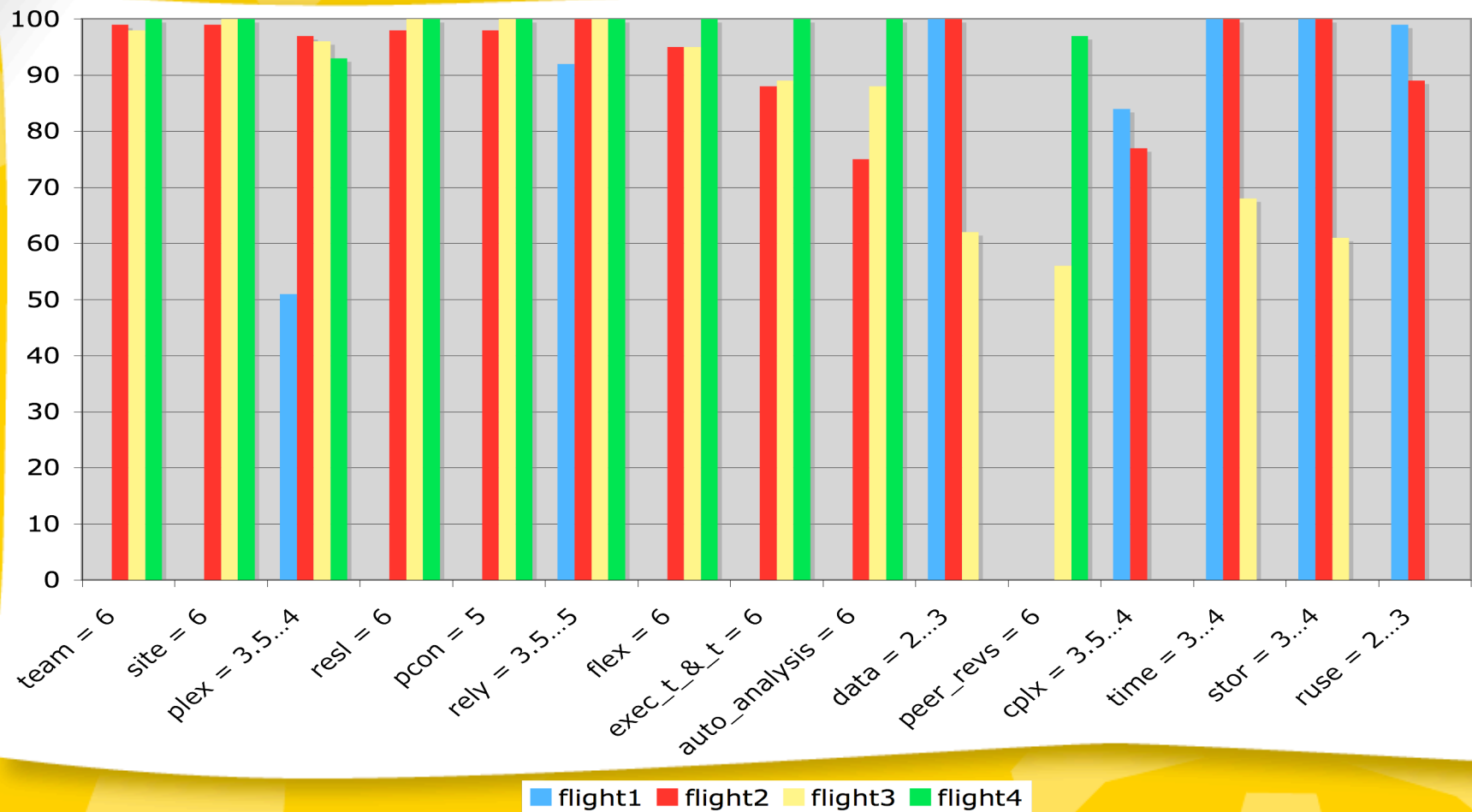
Conclusion

- ⇒ Reuse may be the most beneficial strategy for some projects but can also be worse than doing nothing.
 - The relative merits of software reuse should be evaluated in a project-by-project basis
- ⇒ We presented an approach for such evaluation
 - STAR

Future Work



Future Work





Questions?