Optimizing Apache Ant

What can a Big Software Teach us about Optimization?

Márcio Barros
PPGI - UNIRIO

Who am I?

- Lecturer and researcher at UNIRIO, Rio de Janeiro, Brazil
- Working on Software Engineering for the last 15 years, but with a “dirty” past burning EPROMs and working on operational systems
- UNIRIO stands for “Federal University of Rio de Janeiro State”
- Somewhat confusing when you have the “Federal University of Rio de Janeiro” and the “State University of Rio de Janeiro”!

Research Focus

Search-based Software Engineering

Reformulates SE problems as search problems and applies optimization techniques (most frequently, meta-heuristics) to find good solutions.

Local Search *Simulated Annealing* *CSP*

Genetic Algorithms *Greedy Search*

Research Focus

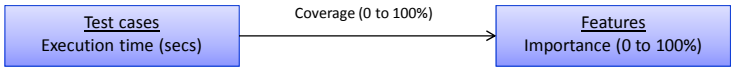
The application of heuristic optimization to SE requires two ingredients …

A representation for the problem along with its solution

A fitness function that evaluates the quality of a solution

Research Focus

Consider the problem of finding a subset of test cases that can run in up to T seconds and cover as much as possible the code that implements some changed features.



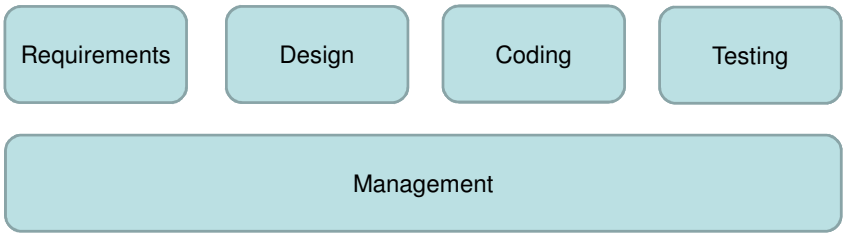
Given a set of test cases $T = \{t_1, t_2, ..., t_N\}$

Maximize $coverage(T) = \sum_{t \in T} \sum_{f \in F} coverage(t, f) * f.importance$

subject to $exectime(T) = \sum_{t \in T} exectime(t) \leq T$

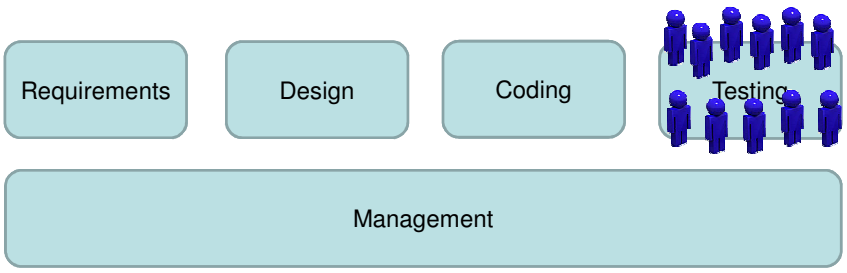
Research Focus

What fields within Software Engineering?



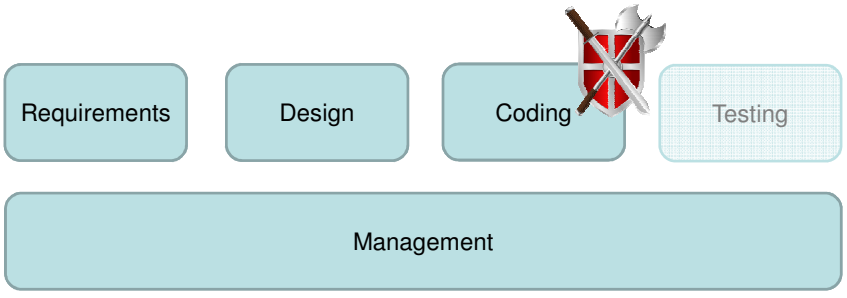
Research Focus

Testing is crowded ...



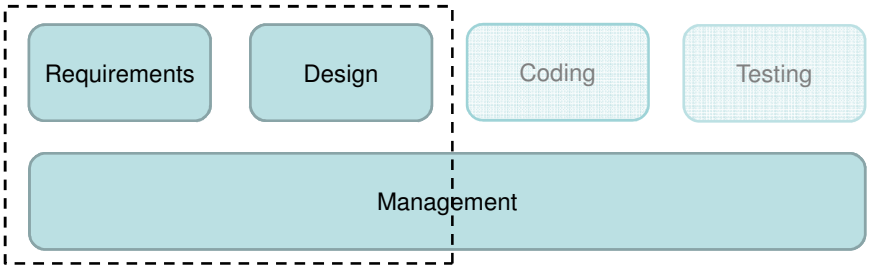
Research Focus

Coding is too hard ... step in with due care.



Research Focus

Focus on the early activities of the traditional life-cycle ...



Today, we focus on design!

Optimizing Apache Ant

Today, I would like to focus on one study we have performed using optimization techniques

We have tried to optimize the design of a big open-source system: Apache Ant


Information and Software Technology

Learning from optimization: A case study with Apache Ant

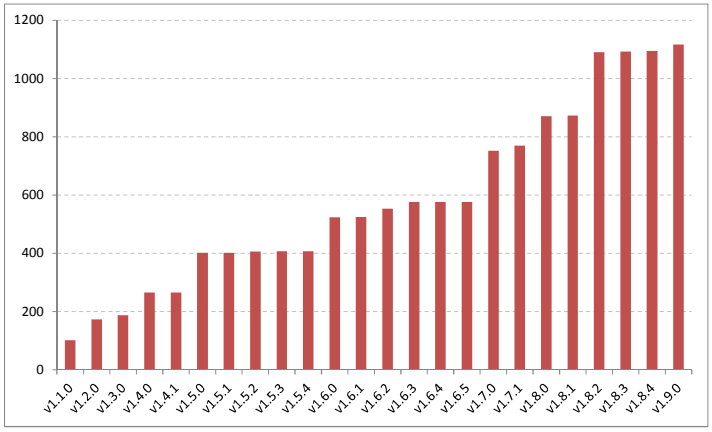
Márcio de Oliveira Barros, Fábio de Almeida Farzat, Guilherme Horta Travassos

Apache Ant

A build automation tool which supports development teams to continuously integrate the results of their software development effort

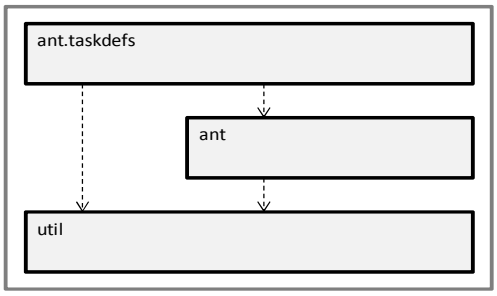
Ant uses a XML files to describe which tasks are required to produce and evaluate the software – and execute these tasks in a proper order.

Is the ANT really big?



Number of classes over different versions

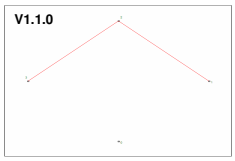
In the beginning, a very simple architecture ...



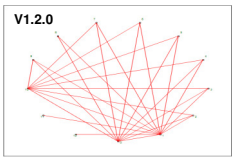
Everything fitted pretty well in just three components

... but evolution has been a quite different story.

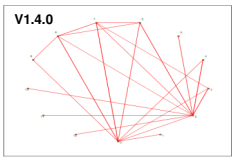
C:102, P: 4 (07/2000)



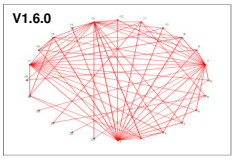
C: 173, P: 13 (10/2000)



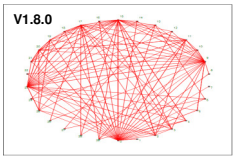
C: 265, P: 13 (09/2001)



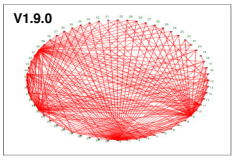
C: 523, P: 24 (12/2003)



C: 870, P: 30 (10/2010)



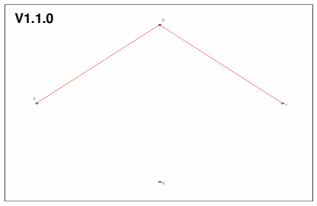
C: 1116, P: 60 (03/2013)



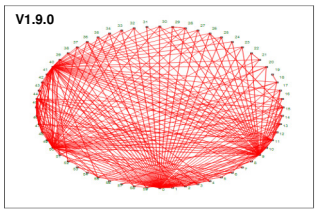
> “Architecture erosion is the process by which a system’s architecture gradually degrades as maintainers make changes to the system that violate the original architectural intents.”

Our fundamental question!

Why does something conceptually as simple as this ...



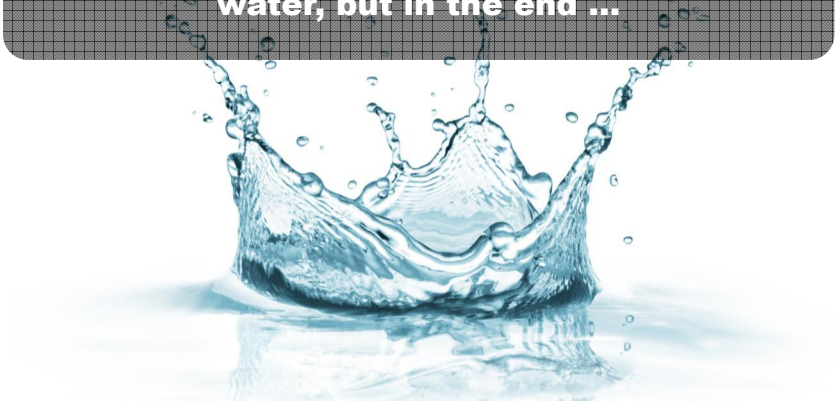
... become such a complicated structure as this?



A good distribution of modules creates a set of building blocks in which each type of change is confined to a well-known set of modules and work assignments may be more easily distributed

What to avoid: shotgun surgery

The change looked like a simple drop of water, but in the end ...



What to avoid: shotgun surgery

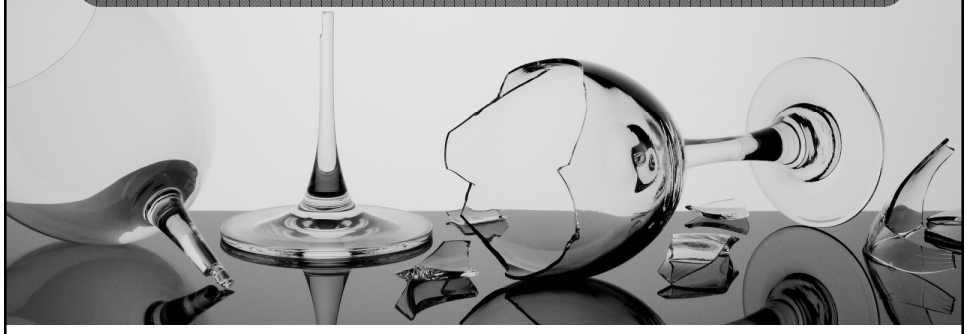
Version	Commits	Single class commits Δ	% Single class commits Δ (%)	Classes per commit ∇	Single package commits Δ	% Single package commits Δ (%)	Packages per commit ∇
1.1.0	172	100	58	2.4 ± 4.6	136	79	1.2 ± 0.5
1.2.0	242	156	64	2.0 ± 2.2	195	81	1.3 ± 0.6
1.3.0	212	135	64	3.1 ± 12.2	176	83	1.4 ± 1.7
1.4.0	348	230	66	2.1 ± 3.1	287	82	1.3 ± 1.1
1.5.0	1704	1087	64	4.6 ± 21.7	1342	79	1.8 ± 3.6
1.6.0	1296	943	73	4.1 ± 36.3	1099	85	1.5 ± 3.9
1.7.0	1238	897	72	2.9 ± 21.8	1034	84	1.5 ± 2.4
1.8.0	611	436	71	2.1 ± 4.3	499	82	1.4 ± 1.6
1.8.2	207	161	78	2.1 ± 6.7	175	85	1.4 ± 1.8
1.9.0	64	36	56	7.7 ± 22.0	47	73	2.5 ± 5.8

~60% of the commits involved a single class (growing trend). Most changes made to the system involved editing a single class. This is an indication that change dispersion was under control.

Shotgun surgery? Not here, it seems.

What to avoid: scattered functionality

Why haven't you implemented that in the same place?



What to avoid: scattered functionality

Version	Commits	Single class commits Δ	% Single class commits Δ (%)	Classes per commit ∇	Single package commits Δ	% Single package commits Δ (%)	Packages per commit ∇
1.1.0	172	100	58	2.4 ± 4.6	136	79	1.2 ± 0.5
1.2.0	242	156	64	2.0 ± 2.2	195	81	1.3 ± 0.6
1.3.0	212	135	64	3.1 ± 12.2	176	83	1.4 ± 1.7
1.4.0	348	230	66	2.1 ± 3.1	287	82	1.3 ± 1.1
1.5.0	1704	1087	64	4.6 ± 21.7	1342	79	1.8 ± 3.6
1.6.0	1296	943	73	4.1 ± 36.3	1099	85	1.5 ± 3.9
1.7.0	1238	897	72	2.9 ± 21.8	1034	84	1.5 ± 2.4
1.8.0	611	436	71	2.1 ± 4.3	499	82	1.4 ± 1.6
1.8.2	207	161	78	2.1 ± 6.7	175	85	1.4 ± 1.8
1.9.0	64	36	56	7.7 ± 22.0	47	73	2.5 ± 5.8

~80% commits involved classes from a single package. Change dispersion is under control: even if multiple classes are being edited as part of a commit, they usually come from the same package.

Scattered?
Where?

What to avoid: disperse coupling and low cohesion

One for all, all for one!

What to avoid: disperse coupling and low cohesion

Version	CBO ∇	AFF ∇	EFF ∇	LCOM ∇	MQ Δ	MF Δ	EVM Δ	CS Δ
1.1.0	0.50	11.8	3.8	0.67	1.79	0.45	−1867	−466
1.2.0	2.23	12.1	7.8	0.74	4.11	0.32	−4011	−308
1.3.0	2.00	21.5	8.6	0.83	4.02	0.50	−4843	−605
1.4.0	2.15	20.2	9.0	0.83	5.83	0.45	−8443	−649
1.5.0	3.33	22.2	13.2	0.81	9.85	0.47	−13,471	−641
1.6.0	3.75	25.4	15.7	0.80	10.94	0.46	−21,519	−896
1.7.0	4.72	33.8	19.9	0.83	11.77	0.41	−39,057	−1346
1.8.0	4.93	38.5	22.1	0.81	12.56	0.42	−49,381	−1646
1.8.2	4.58	26.1	17.5	0.79	21.37	0.36	−51,615	−874
1.9.0	4.65	26.4	17.7	0.79	22.05	0.37	−49,639	−827

Strongly correlated measures.

Coupling is growing and that is bad.

Or at least it should be?

Cohesion is stable.

But these guys are among many other villains ...

Shotgun surgery

Scattered functionality

Disperse coupling

Low cohesion

Feature envy

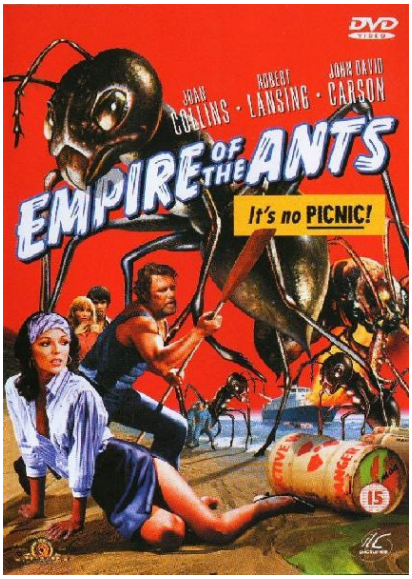
God classes

Unused interfaces

Connector envy

and it goes on and on …

Together, these coding problems may turn your software into a nightmare



Mixed signs of problems and stability ...

... but, in the end, the architecture is growing increasingly complex!

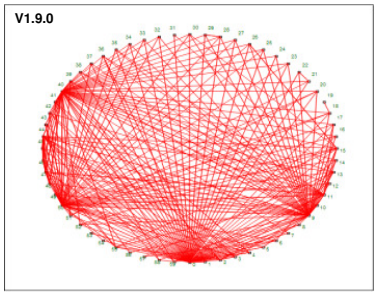
SBSE to the Rescue

Can search-based software module clustering aid in restoring the original architecture?

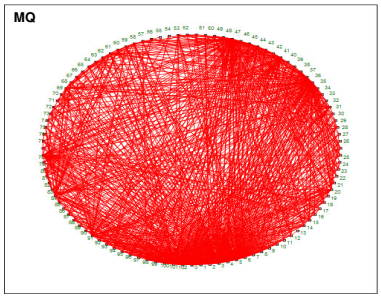
Unexpected results with our current model

The selected metrics were strongly improved, but the design is quite different from the original architecture.

The original architecture (MQ ~21)



Optimized architecture MQ: ~100



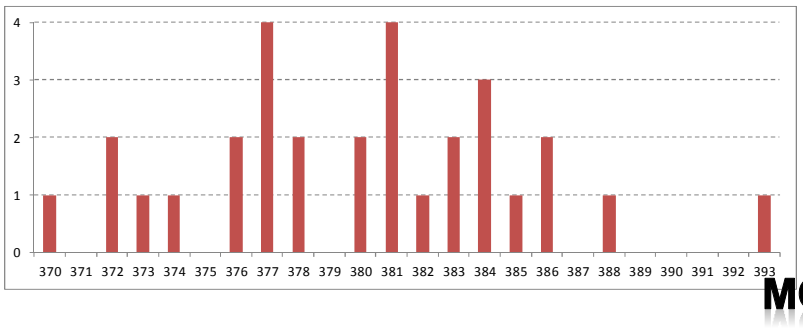
Unexpected results with our current model

Version	CBO ∇	AFF ∇	EFF ∇	LCOM ∇	MQ Δ	EVM Δ
Original v1.9.0	4.65	26.4	17.7	0.79	22.05	−49,639
EVM-optimized	4.48 ± 0.14	5.64 ± 0.19	5.88 ± 0.06	0.41 ± 0.01	73.62 ± 1.14	773.9 ± 7.89
MQ-optimized	7.78 ± 0.15	10.49 ± 0.18	8.68 ± 0.09	0.46 ± 0.01	101.63 ± 1.20	289.1 ± 15.27

Version	Single package commits Δ	% Single package commits Δ	Packages per commit ∇
Original v1.9.0	47	73	2.5 ± 5.8
EVM-optimized	38.5 ± 1.1	60	5.3 ± 15.1
MQ-optimized	37.5 ± 0.5	59	5.6 ± 16.3

Results are worse for 3 out of 4 indirect measures.

The scape route ...



Many more packages containing much less classes.

The scape route ...

- Optimization finds unexpected ways to improve the fitness function
- Like “wild code”, optimization does what the researcher tell it to do, not necessarily what s/he wants!

So, what can a big software teach us about optimization?

It taught us that SBSE is also a learning tool

So, what can a big software teach us about optimization?

- Optimization take our theories and the models that describe them to extreme … and it allows us to see their behavior on such scenarios (probably for the first time)
- At least for software clustering, we need models and metrics that better reflect developer's intention in large software systems
- The dogma of increasing cohesion & reducing coupling is being questioned (an not for the first time, e.g. Anquetil & Laval, 2011)

So, what can a big software teach us about optimization?

- We need to find ways to introduce multiple user perceptions into the optimization process (there seems to be a lot of gut feeling here!)
- Some steps toward this goal have been made (Hall et al., 2012; Bavota et al., 2012), but we need to take problems to a level on which developers can contribute and let optimization fill the gaps
- Learn from developers using automated learning processes and drive optimization on these uncovered trends
- We need new models … more and more evidence about that is coming soon!

That is what we have learned.

Thank you!