The Lost Tales of Platform Design

Wisdom from the Masters, learned the Hard Way

Julien Simon
julien@julien.org
@julsimon

Who am I ?

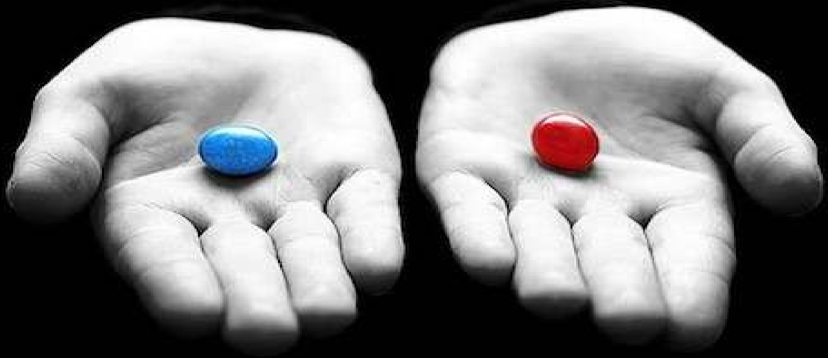
- Software Engineer
- 20+ years in R&D teams, from smartcards to web platforms
- VP Eng/CTO @ Digiplug, Pixmania, Criteo, Aldebaran Robotics, Viadeo
- Now working for Amazon Web Services

Agenda

- The Only Thing that Matters
- A Method to the Madness
- Infrastructure
- Architecture
- Code Hygiene
- Optimization
- The Bleeding Edge
- Summing things up

The only thing that matters

no, not drugs

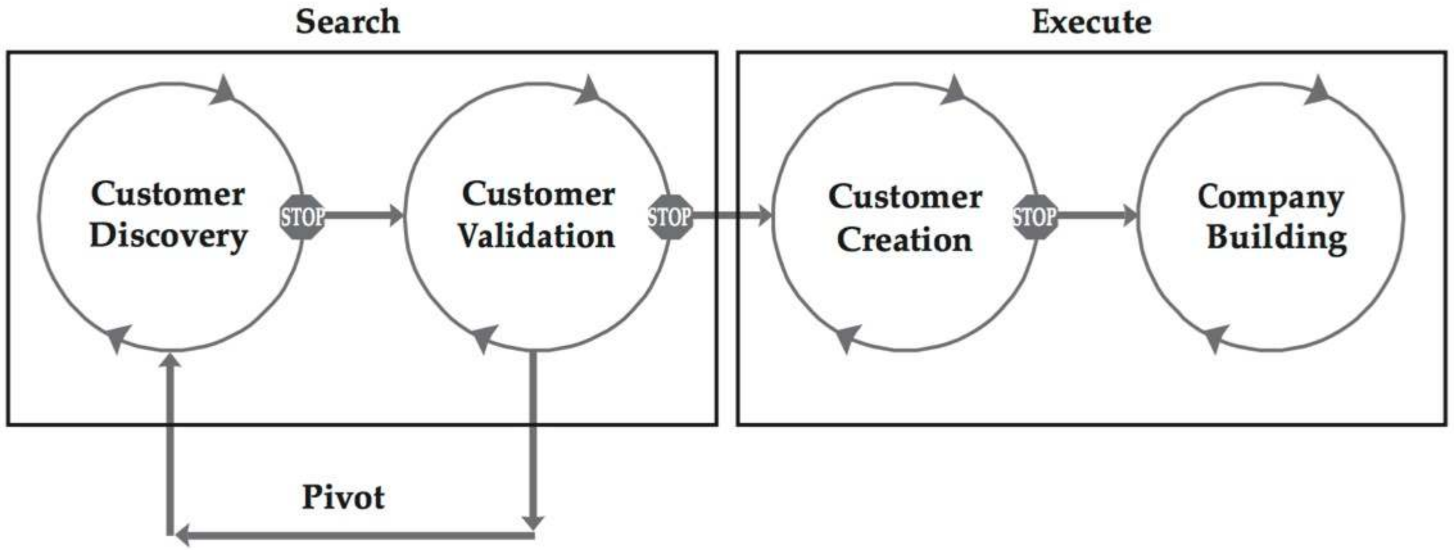


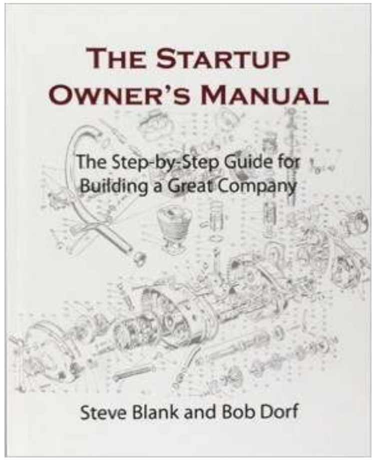
What Founders & Investors obsess about

“Do whatever is required to get to product/market fit. Including changing out people, rewriting your product, moving into a different market, telling customers no when you don't want to, telling customers yes when you don't want to, raising that fourth round of highly dilutive venture capital - whatever is required. When you get right down to it, you can ignore almost everything else”

Marc Andreessen, June 2007
http://web.archive.org/web/20070701074943/http://blog.pmarca.com/2007/06/the-pmarca-gu-2.html

The Startup Owner's Manual - Steve Blank & Bob Dorf (2012)





THE STARTUP OWNER'S MANUAL
The Step-by-Step Guide for Building a Great Company
Steve Blank and Bob Dorf

“I’m a dev, that’s not my problem”

EXCUSE ME?

Are you a pro, or just playing with computers?

What do you think you’re getting paid for?

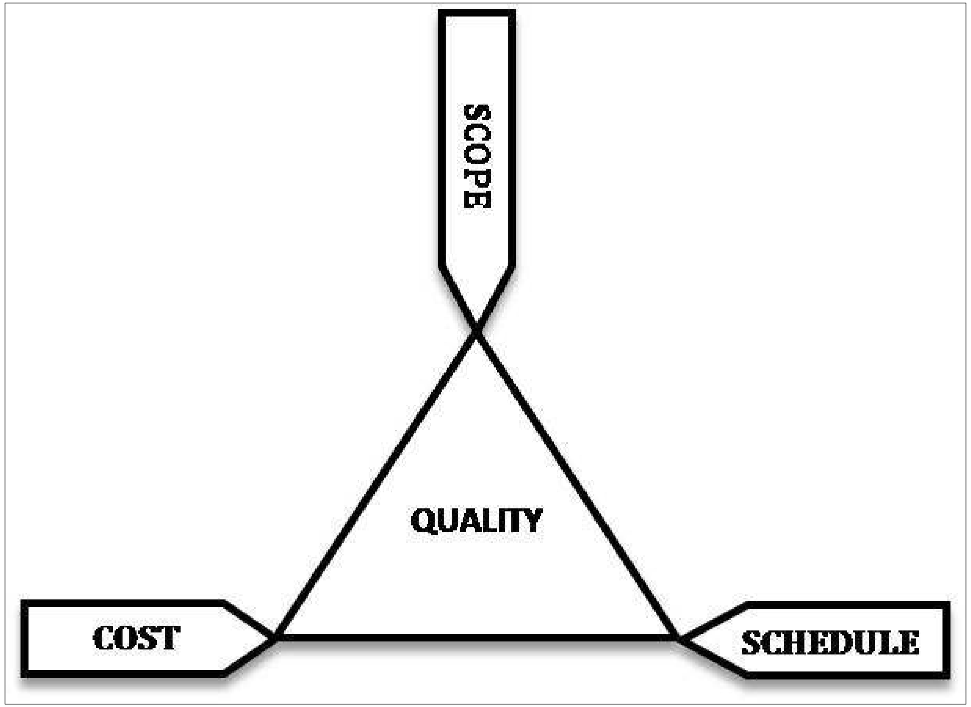
What do you think happens when the company runs out of cash?

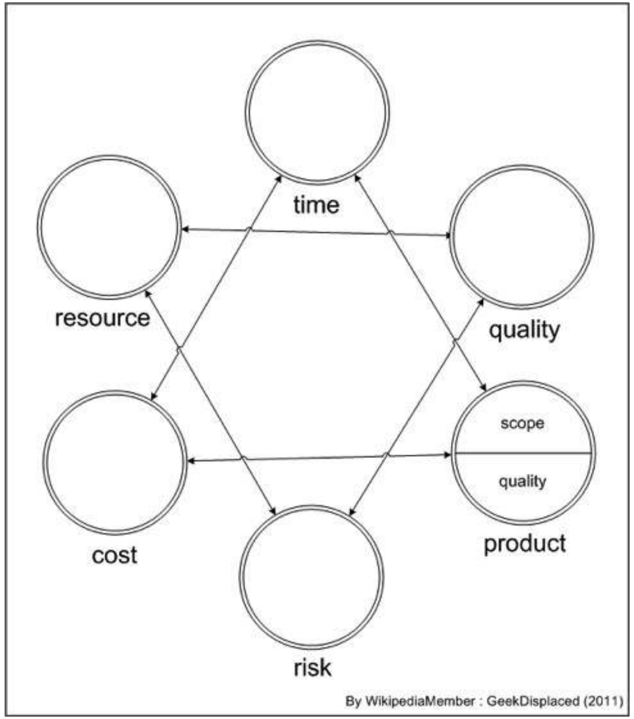
Do you want to work for shit companies or great ones?

A Method to the Madness

The Trilemma of Project Management

aka The Iron Triangle (Martin Barnes, 1969)

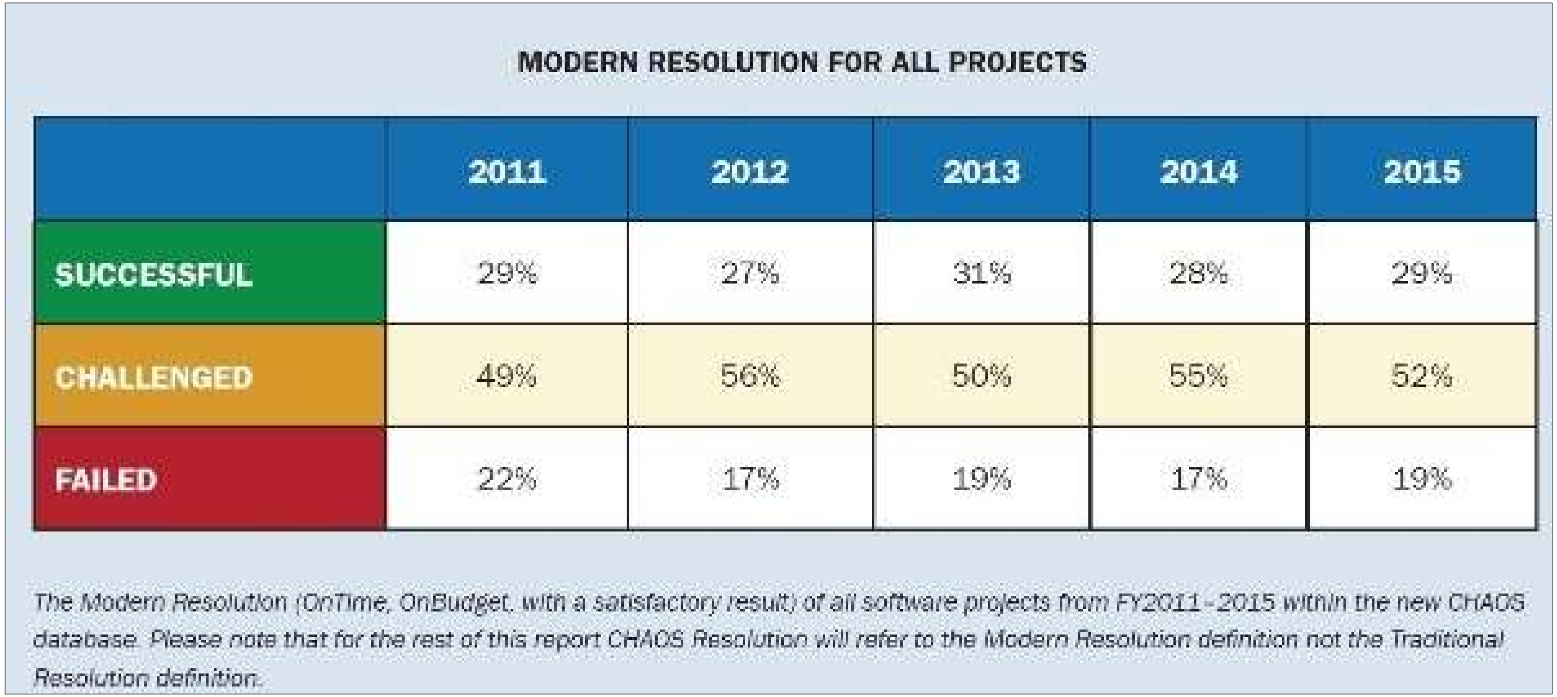




https://en.wikipedia.org/wiki/Project_management_triangle
https://www.pmi.org/pmbok-guide-standards

Less than 1/3 of projects are successful

and it's not getting better



MODERN RESOLUTION FOR ALL PROJECTS

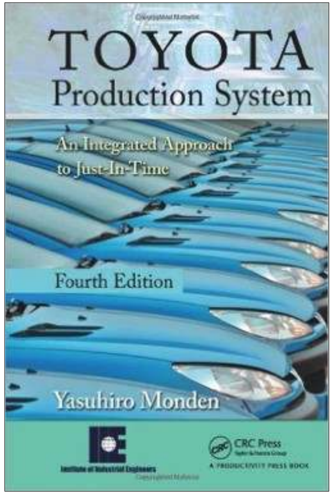
	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011-2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.

https://www.infoq.com/articles/standish-chaos-2015

Kaizen (1950)

改 善


TOYOTA
Production System
An Integrated Approach to Just-In-Time
Fourth Edition
Yasuhiro Monden
Institute of Industrial Engineers
CRC Press
A PRODUCTIVITY PRESS BOOK

Reduce waste

Spend no money
Add no people
Add no space

https://en.wikipedia.org/wiki/Kaizen
https://en.wikipedia.org/wiki/Toyota_Production_System

Boyd's Law (1950)

- Boyd was a US Air Force Colonel, trying to understand why the highly superior MIG-15 was consistently losing dogfights to the F-86
- Answer: the hydraulic controls of the F-86 allowed its pilot to repeatedly observe, orient, plan and act with less effort than the MIG-15 pilot
- Hence, Boyd's Law: in analyzing complexity, fast iteration almost always produces better results than in-depth analysis

"A Better Path to Enterprise Architectures" (2006) https://msdn.microsoft.com/en-us/library/aa479371.aspx

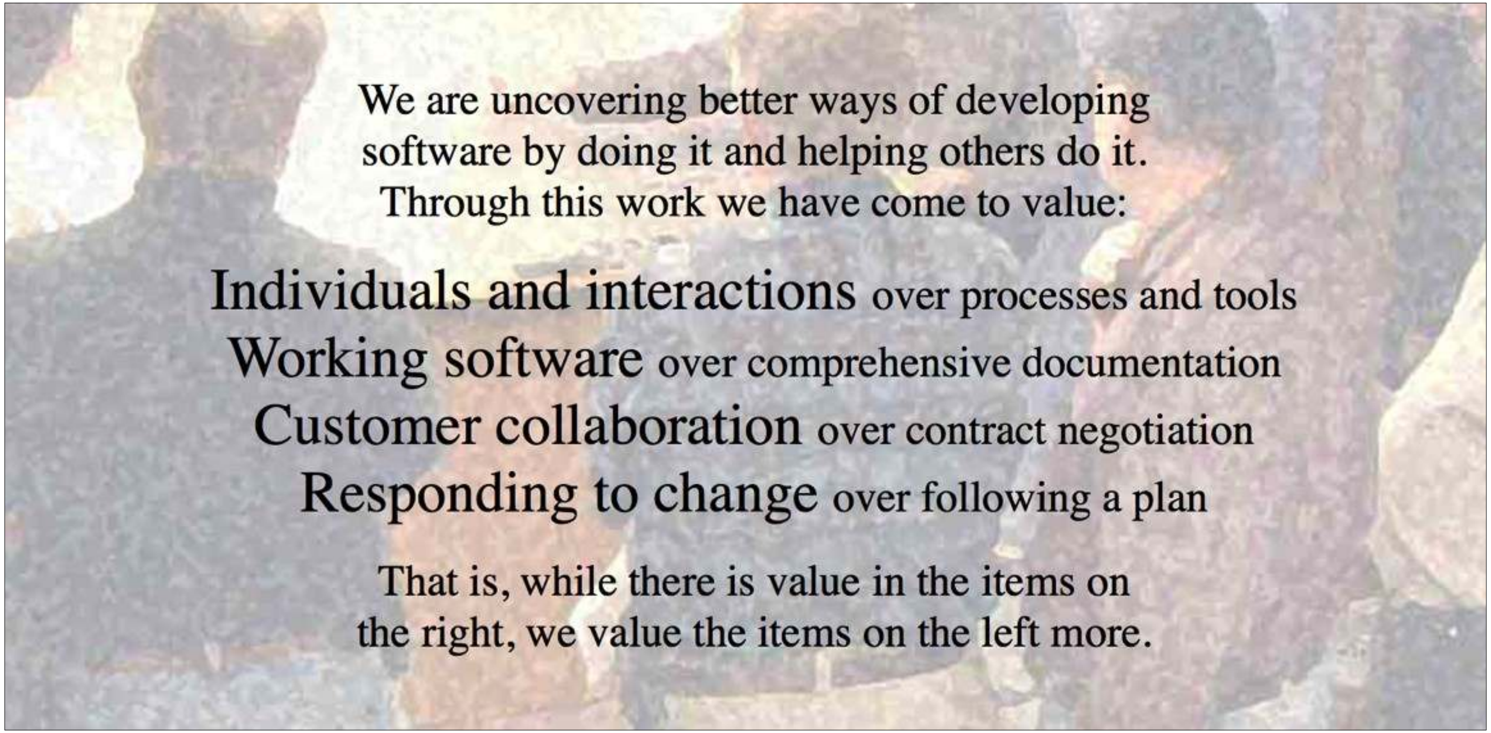
No Silver Bullet — Essence and Acciden of Software Engineering (Fred Brooks, 1

"The essence of a software entity is a construct of interlocking c data sets, relationships among data items, algorithms, and invocati functions […] I believe *the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation […] If this is true, building software will always be hard. There is inherently no silver bullet."*

- *Exploiting the mass market to avoid constructing what can be bought.*
- *Using rapid prototyping as part of a planned iteration in establishing software requirements.*
- *Growing software organically, adding more and more function to systems as they are run, used, and tested.*
- *Identifying and developing the great conceptual designers of the rising generation*

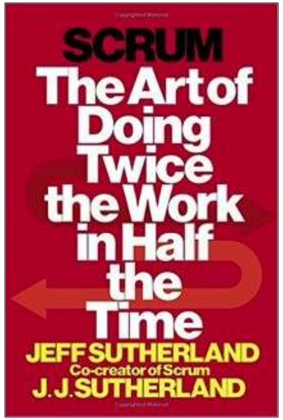
http://faculty.salisbury.edu/~xswang/Research/Papers/SERelated/no-silver-bullet.pdf

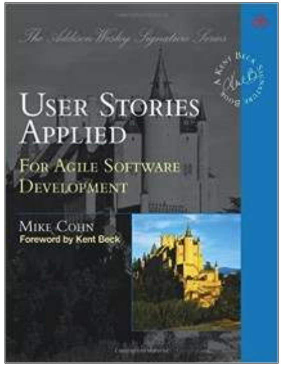
The Agile Manifesto & Scrum (2001)


We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

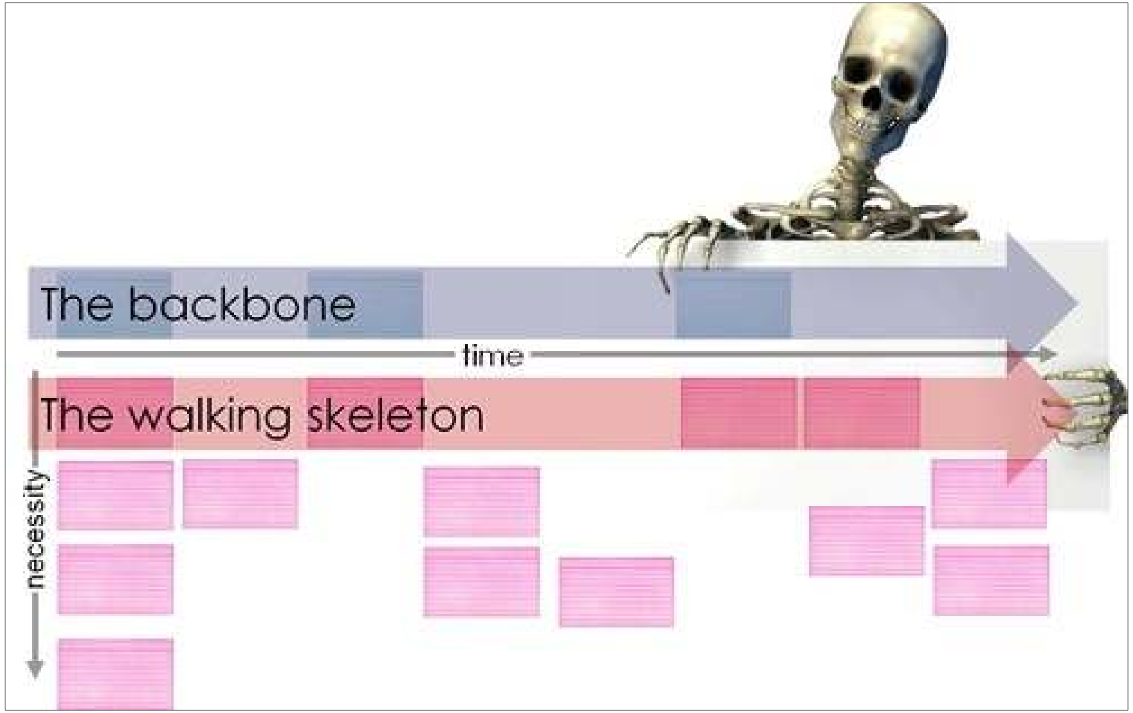
That is, while there is value in the items on the right, we value the items on the left more.

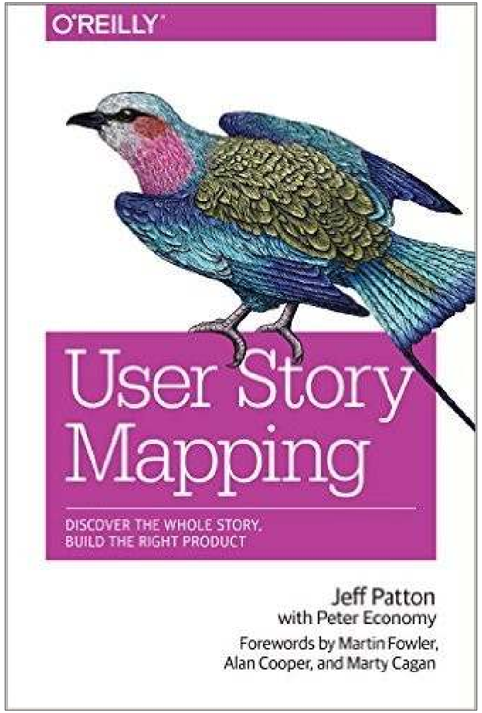



The Addison-Wesley Signature Series
User Stories Applied
For Agile Software Development
Mike Cohn
Foreword by Kent Beck
A Kent Beck Signature Book

http://agilemanifesto.org

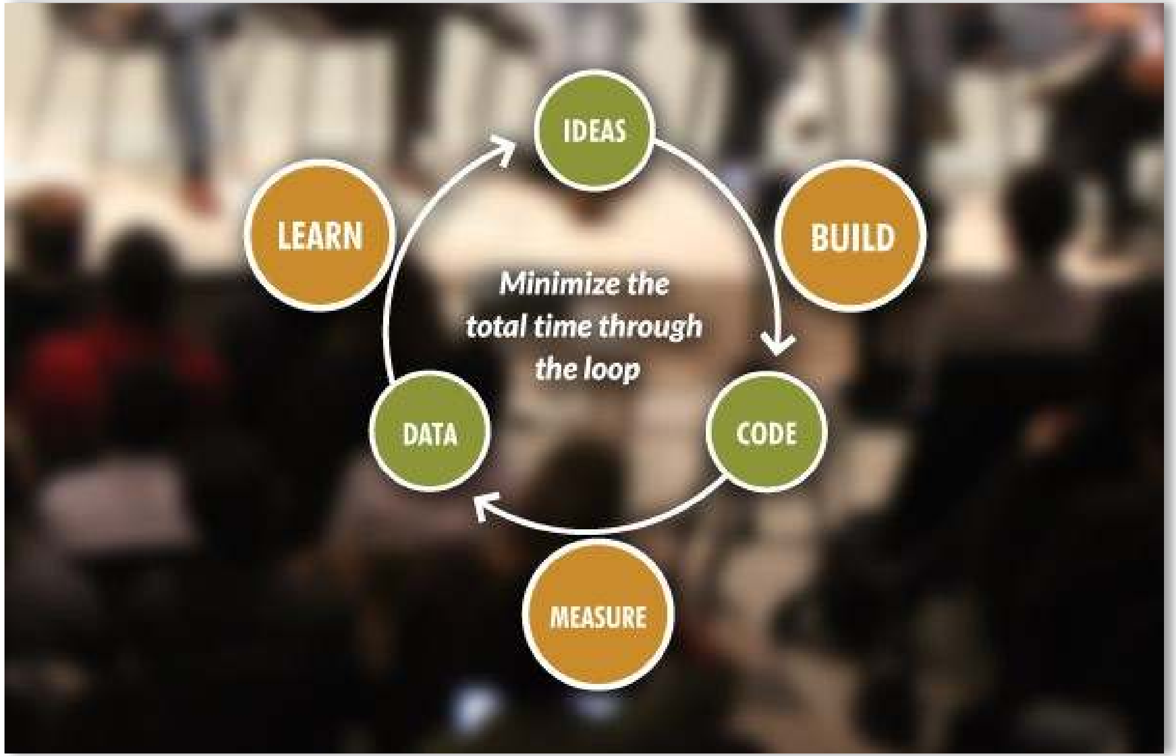
Story Mapping (Jeff Patton, 2005)

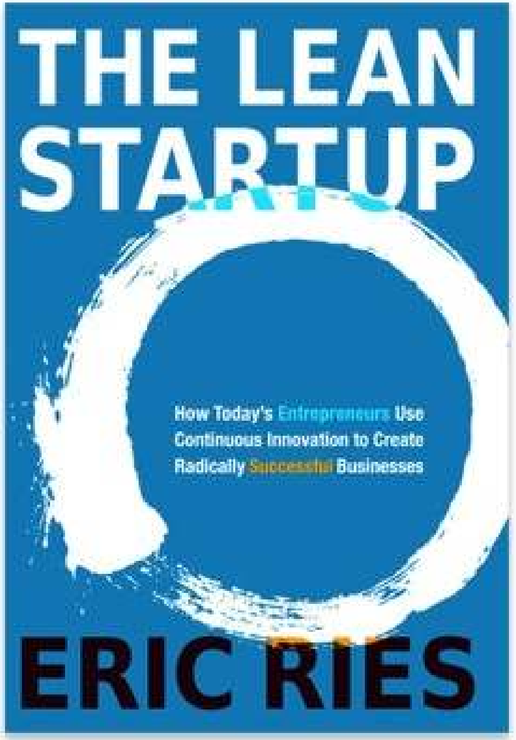



O'REILLY
User Story Mapping
DISCOVER THE WHOLE STORY, BUILD THE RIGHT PRODUCT
Jeff Patton
with Peter Economy
Forewords by Martin Fowler, Alan Cooper, and Marty Cagan

http://jpattonassociates.com/wp-content/uploads/2015/01/how_you_slice_it.pdf
http://jpattonassociates.com/the-new-backlog/

The Lean Startup (Eric Ries, 2011)

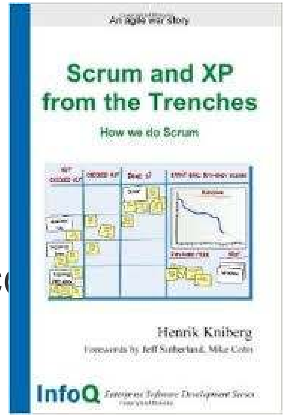



THE LEAN STARTUP
How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses
ERIC RIES

http://theleanstartup.com

Caveats

- Agile has become a cult, like ISO9001 30 years ago.
- Be a practitioner, not a priest.
- Yes, it's OK to adapt Scrum/Kanban to your own c
  Whatever works.
- The Agile Manifesto is the light in the dark.
  Stay on the path, you'll be fine.
- Lean, MVP, walking skeleton: fine, but make sure you eventually deliver something consistent. Sum of demos and PoCs != Product
- « Fail fast », « move fast, break stuff », « trial and error »: fine too, but make sure you have tests and metrics, or how will you know you failed?
- Which brings me to…



An agile war story

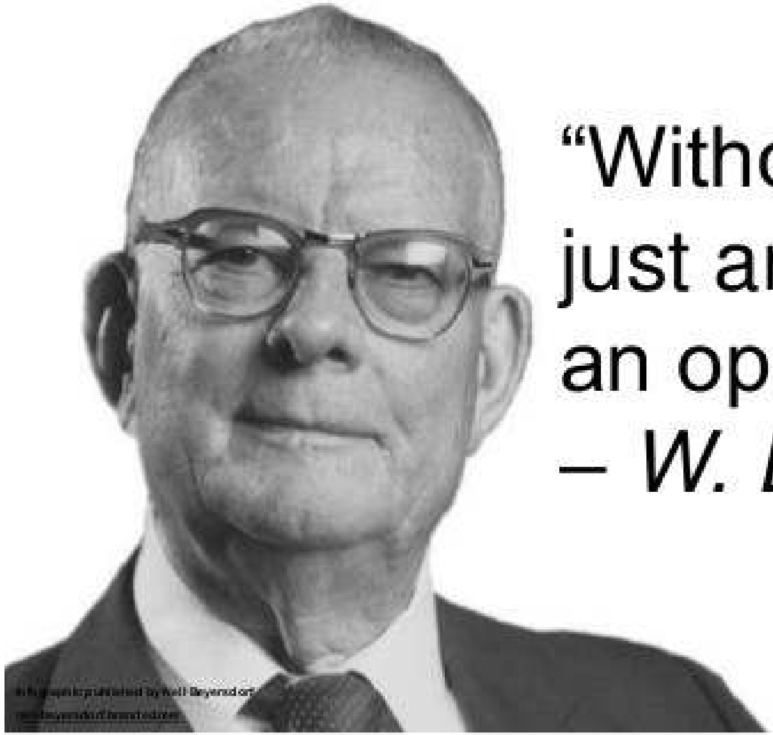
Scrum and XP from the Trenches

How we do Scrum

Henrik Kniberg

Forewords by Jeff Sutherland, Mike Cohn

InfoQ Enterprise Software Development Series



“Without data you’re just another person with an opinion.”
– *W. Edwards Deming*

Opinions are like assholes.
Everybody has one.

In summary, you are expected to…

- Deliver new, working code at the drop of a hat
- Build it AND run it (DevOps, remember?)
- Iterate, learn & adapt as fast as possible
- Manage all necessary data for company-wide decisions
- Preserve cash while doing it: survival starts with not spending foolishly
- Be prepared to scale non-stop when you’re told to

Infrastructure

Get it out of the way

Ready for 30 seconds of infrastructure bliss?

How about the real story?

Meet the Horsemen of Infrapocalypse ©

Buy

Build

Run

Disaster recovery

Buying physical infrastructure

Building physical infrastructure

Personal collection

Running, scaling, fixing

Personal collection

Disaster Recovery (or not)

Verizon facilities (Nov'12, Sandy in NYC)

Personal collection

IBM datacenter (Dec'11, Tokyo)

Do you see the light?

Reminder:

- Deliver new, working code at the drop of a hat
- Build it AND run it (DevOps, remember?)
- Iterate, learn & adapt as fast as possible
- Manage all necessary data for company-wide decisions
- Preserve cash while doing it: survival starts with not spending foolishly
- Be prepared to scale like Hell when you're told to

Ask yourself:
does physical infrastructure help with these goals?
Or does it stand in the way?

Yeah, thought so. The Cloud, then.

Minimum Viable Infrastructure

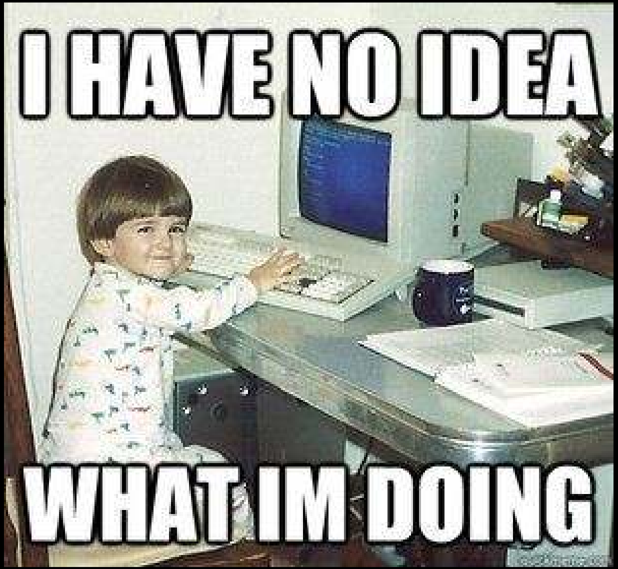
- Infrastructure should be iterative as well

- Early days
  - Take the fastest route to customer discovery
  - PaaS platforms rock: Heroku, Elastic Beanstalk, Convox, etc.

- As you grow
  - Immutable infrastructure (and Docker isn't the only way)
  - No plumbing! Use Managed Services as much as you can
  - Automate everything
    - Infrastructure Deployment: CloudFormation, Terraform, etc.
    - Application Deployment: Chef, Puppet, etc.
  - Scale-out

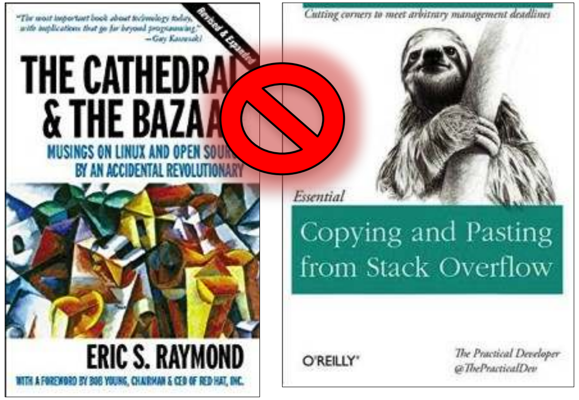
http://www.slideshare.net/nzoschke/minimum-viable-infrastructure

Architecture

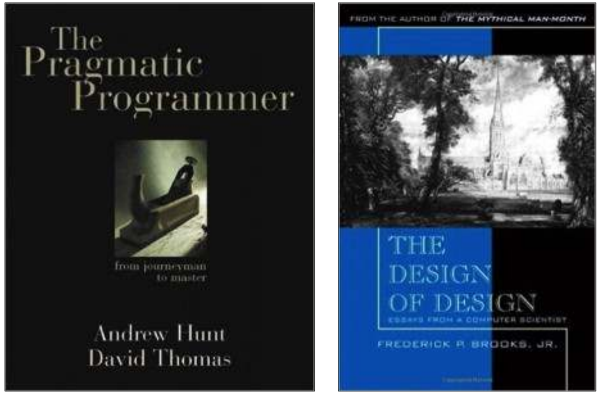


A generation lost in the Bazaar (PHK, 2011)

That is the sorry reality of the bazaar Raymond praised in his book: a pile of old festering hacks, endlessly copied and pasted by a clueless generation of IT "professionals" who wouldn't recognize sound IT architecture if you hit them over the head with it.


THE CATHEDRAL & THE BAZAAR
MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY
ERIC S. RAYMOND
Cutting corners to meet arbitrary management deadlines
Essential
Copying and Pasting from Stack Overflow
O'REILLY
The Practical Developer @ThePracticalDev

"Quality happens only when someone is responsible for it"

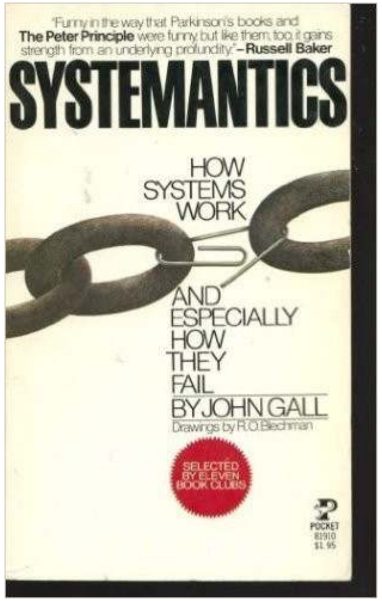

The Pragmatic Programmer
from journeyman to master
Andrew Hunt
David Thomas
FROM THE AUTHOR OF THE MYTHICAL MAN-MONTH
THE DESIGN OF DESIGN
ESSAYS FROM A COMPUTER SCIENTIST
FREDERICK P. BROOKS, JR.

https://en.wikipedia.org/wiki/Poul-Henning_Kamp
http://queue.acm.org/detail.cfm?id=2349257

Gall's Law (1975)

A complex system that works is invariably found to have evolved from a simple system that worked.

A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over with a working simple system.

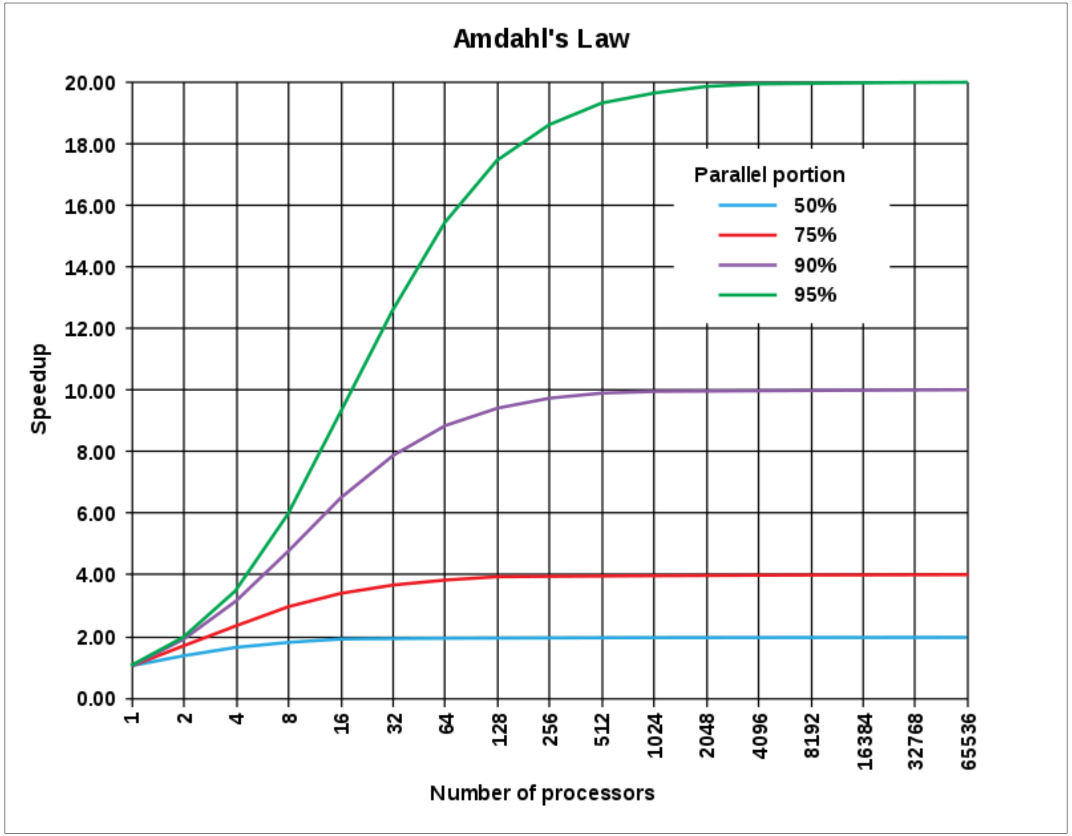

"Funny in the way that Parkinson's books and The Peter Principle were funny but like them, too, it gains strength from an underlying profundity." –Russell Baker
SYSTEMANTICS
HOW SYSTEMS WORK
AND ESPECIALLY HOW THEY FAIL
BY JOHN GALL
Drawings by R.O. Blechman
SELECTED BY ELEVEN BOOK CLUBS
POCKET 81910 $1.95

https://en.wikipedia.org/wiki/John_Gall_(author)

Amdahl's Law (1967)

- Theoretical speedup of the execution of a task - at fixed workload - that can be expected of a system whose resources are improved.

- In other words: hardware parallelism only helps as far as software can be parallelized. Moore's Law doesn't come for free!



https://en.wikipedia.org/wiki/Amdahl's_law

“You are not paid to write code” (2016)

Many people grumble or reject the notion of using proven tools or techniques. It’s boring. It requires investing time to learn at the expense of shipping code. It doesn’t do this one thing that we need it to do. It won’t work for us. For some reason—and I continue to be completely baffled by this—everyone sees their situation as a unique snowflake despite the fact that a million other people have probably done the same thing […]

I was able to come to terms with this once I internalized something a colleague once said: you are not paid to write code. You have never been paid to write code. In fact, code is a nasty byproduct of being a software engineer.

http://bravenewgeek.com/you-are-not-paid-to-write-code/

I’ll say it again: don’t write code unless you must

- Every single line of code is technical debt
- Design Patterns: only if they make sense. Don’t play with yourself.
- What’s your self-delusional story for not using:
  - Off-the-shelf libs & tools
    - Standard language libs, Boost, Guava, etc.
    - Logging, parsing, math, crypto, etc.
  - SaaS platforms
    - Authentication, SSO
    - Tags, trackers, A/B testing, dashboards
    - Complex but non-core business processes: media transcoding, document processing, etc.

Monitoring is a first-class citizen

- Design for monitoring from day 1.
- Don’t send monitoring information by email. No one reads it.
- Logging to local files is not enough and may create more problems
  - Centralize log storage and processing (ELK, Splunk, etc.)
  - If something is really urgent, it shouldn’t be written to a file!

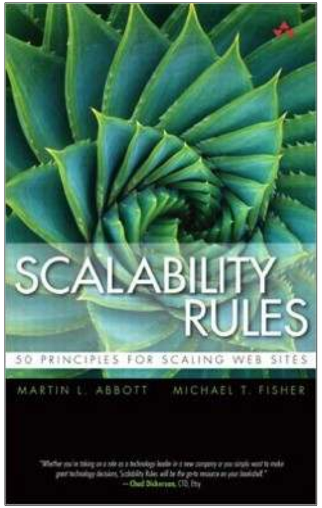
- Monitor all services carefully
  - Inside view and outside view (Catchpoint, PagerDuty)
  - Don’t forget to monitor 3rd party services…
- Real-time technical & business metrics are priceless (Graphite, Kibana)

Do it to them before they do it to you

- Make sure you understand how things fail (periodic testing)
- Make sure you understand why and how they failed (post mortems)
- Implement circuit breakers and feature toggles to limit blast radius
- Find out how your service behaves in degraded mode
- Break stuff, see what happens. Most of your assumptions will be wrong.

Minimum Viable Architecture

- Early days: take the fastest route to customer discovery. No points for style, but you should be aware of the shortcuts.
- As you grow: APIs and microservices, but there is no silver bullet ☺
  - State management
  - Dependency Hell
  - Impedance mismatch between services and backends
  - Deployment & monitoring
  - Authentication across services
  - Etc.
- Good news: scalability is almost free when your design is clean



http://www.slideshare.net/RandyShoup/minimum-viable-architecture-good-enough-is-good-enough-in-a-startup

Code Hygiene

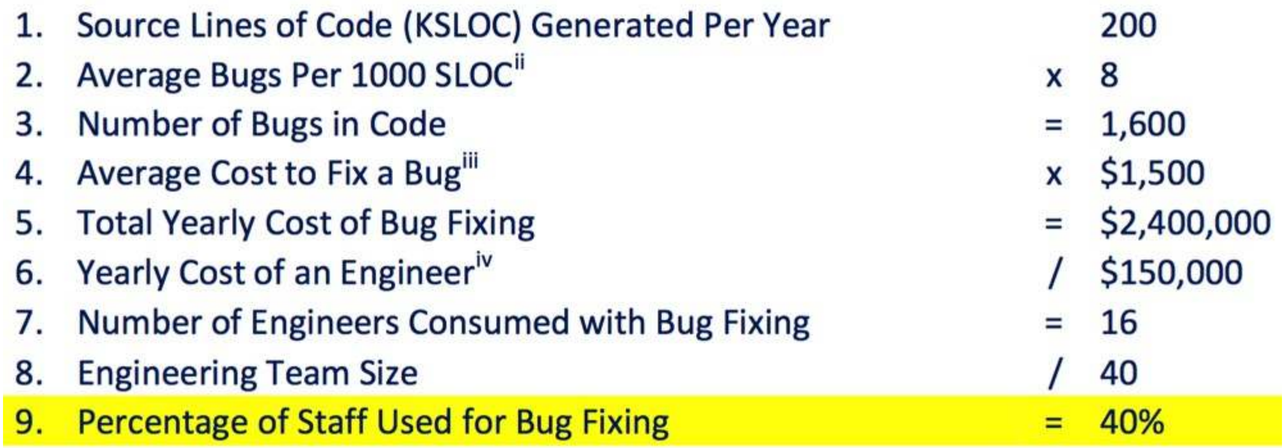
Coding vs Debugging vs WTF

- How much of your time is spent
  - working on product features?
  - investigating or fixing bugs?
  - doing something else?

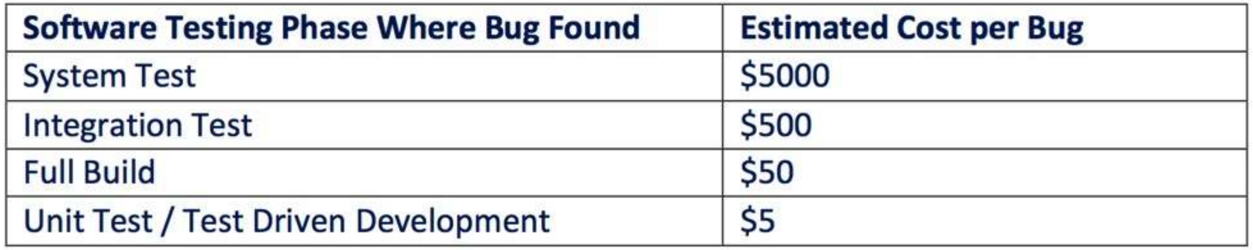
- From my experience: 25% / 50% / 25%

- You only get a day’s worth of coding per week. Make it count.

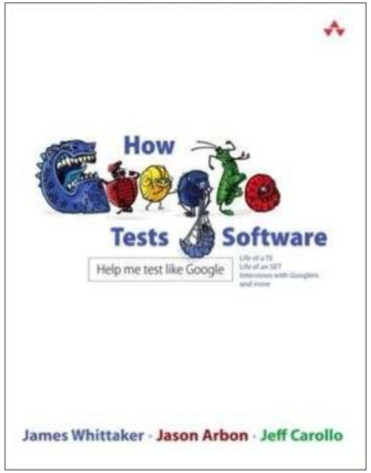
Half your time is spent fixing bugs



1.	Source Lines of Code (KSLOC) Generated Per Year		200
2.	Average Bugs Per 1000 SLOC[ii]	x	8
3.	Number of Bugs in Code	=	1,600
4.	Average Cost to Fix a Bug[iii]	x	$1,500
5.	Total Yearly Cost of Bug Fixing	=	$2,400,000
6.	Yearly Cost of an Engineer[iv]	/	$150,000
7.	Number of Engineers Consumed with Bug Fixing	=	16
8.	Engineering Team Size	/	40
9.	Percentage of Staff Used for Bug Fixing	=	40%



Software Testing Phase Where Bug Found	Estimated Cost per Bug
System Test	$5000
Integration Test	$500
Full Build	$50
Unit Test / Test Driven Development	$5


How Google Tests Software
Help me test like Google
James Whittaker · Jason Arbon · Jeff Carollo

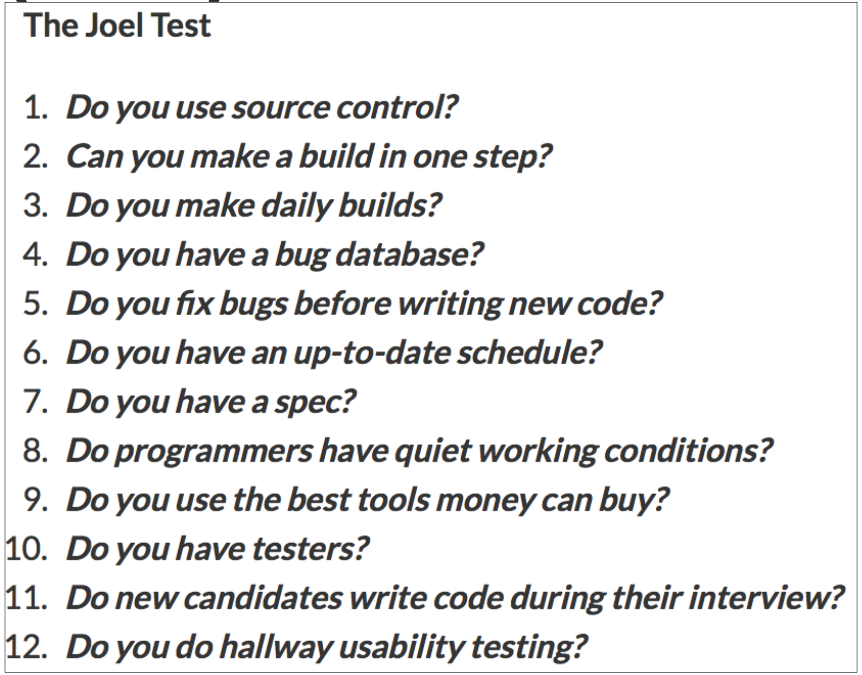
http://www.newelectronics.co.uk/article-images/65147%5CVector_PDF.pdf

Spending less time & money on bugfixing

- Write less code
- Find bugs faster and sooner
  - Deploy the best CI/CD you can get
  - And don't go building it! Lots of SaaS options: Travis, CircleCI, etc.
- Fix bugs faster
  - Deploy the best real-time monitoring you can get
  - If you have budget for APM tools like New Relic or Dynatrace, spend it!
- Don't fix all bugs: "*Fixing bugs is only important when the value of having the bug fixed exceeds the cost of the fixing it*" - Joel Spolsky

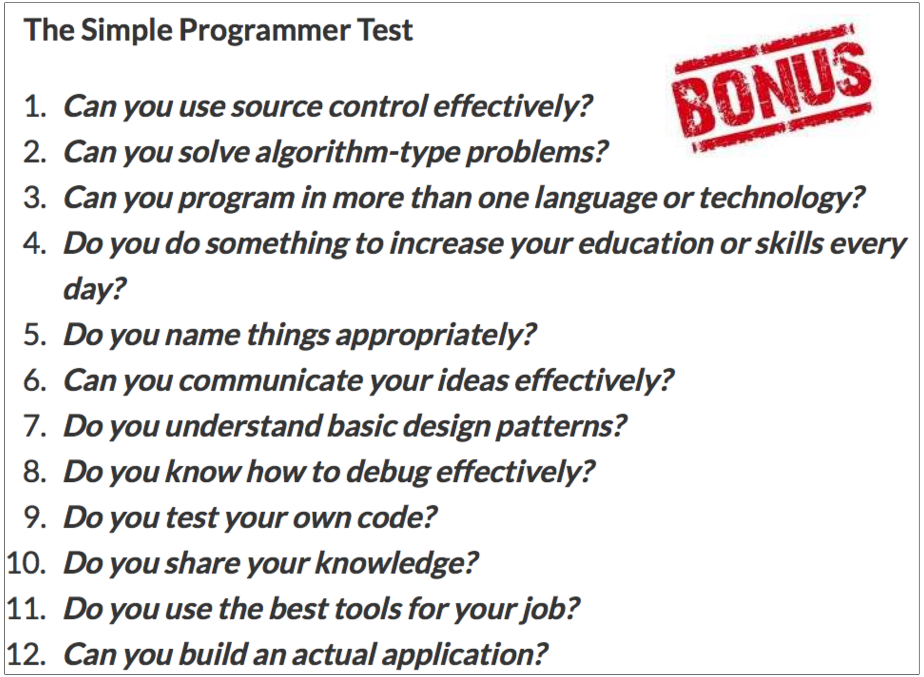
http://www.joelonsoftware.com/articles/fog0000000014.html

The Joel Test – 12 Steps to Better Code (2001)



The Joel Test

1. *Do you use source control?*
2. *Can you make a build in one step?*
3. *Do you make daily builds?*
4. *Do you have a bug database?*
5. *Do you fix bugs before writing new code?*
6. *Do you have an up-to-date schedule?*
7. *Do you have a spec?*
8. *Do programmers have quiet working conditions?*
9. *Do you use the best tools money can buy?*
10. *Do you have testers?*
11. *Do new candidates write code during their interview?*
12. *Do you do hallway usability testing?*



The Simple Programmer Test

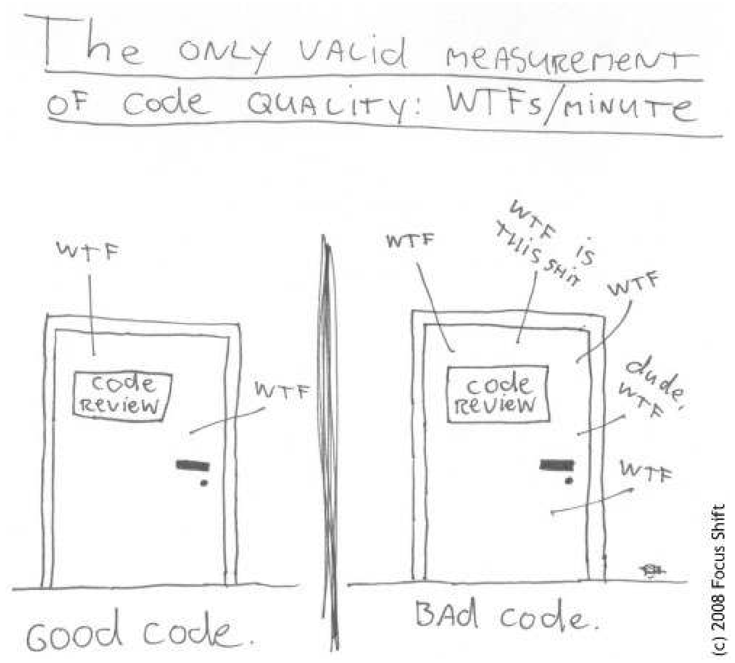
BONUS

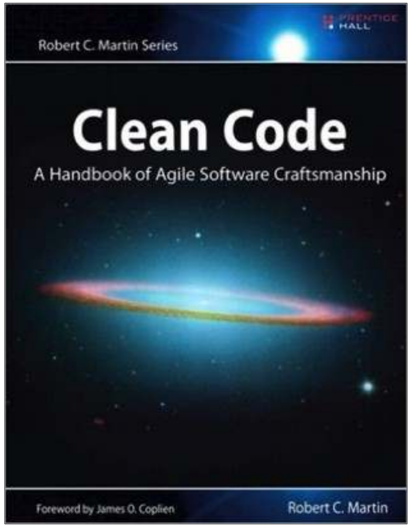
1. *Can you use source control effectively?*
2. *Can you solve algorithm-type problems?*
3. *Can you program in more than one language or technology?*
4. *Do you do something to increase your education or skills every day?*
5. *Do you name things appropriately?*
6. *Can you communicate your ideas effectively?*
7. *Do you understand basic design patterns?*
8. *Do you know how to debug effectively?*
9. *Do you test your own code?*
10. *Do you share your knowledge?*
11. *Do you use the best tools for your job?*
12. *Can you build an actual application?*

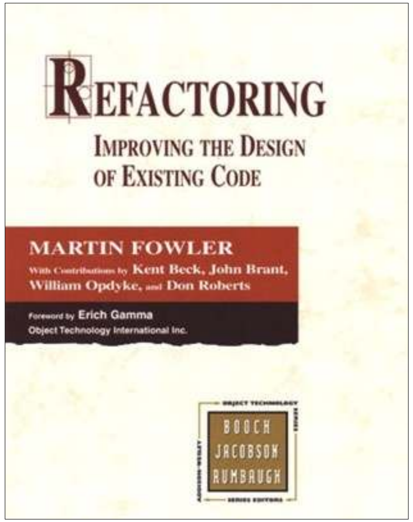
Your real score is the one I'd find if I audited you ;)

http://www.joelonsoftware.com/articles/fog0000000043.html
https://simpleprogrammer.com/2015/02/16/joel-test-programmers-simple-programmer-test/

Code smell




Robert C. Martin Series
PRENTICE HALL
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin


Refactoring
Improving the Design of Existing Code
Martin Fowler
With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.
Booch Jacobson Rumbaugh

http://martinfowler.com/bliki/CodeSmell.html
https://blog.codinghorror.com/code-smells/

More articles

- “How to deploy software” (2016)
  https://zachholman.com/posts/deploying-software
- “How terrible code gets written by perfectly sane people“ (2016)
  https://techbeacon.com/how-terrible-code-gets-written-perfectly-sane-people

Optimization

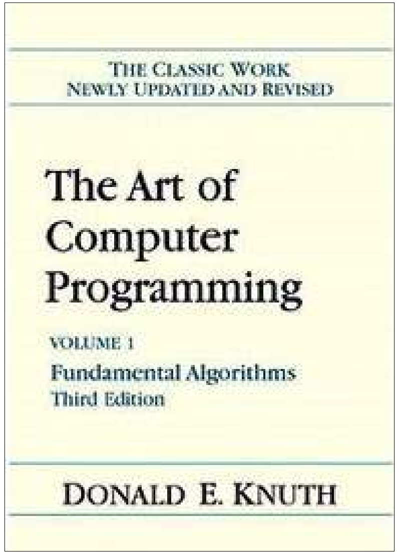
Playing with yourself again?

(thank me for not adding a picture here)

Premature optimization is the root of all evil (Knuth, 1974)

Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered.

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%

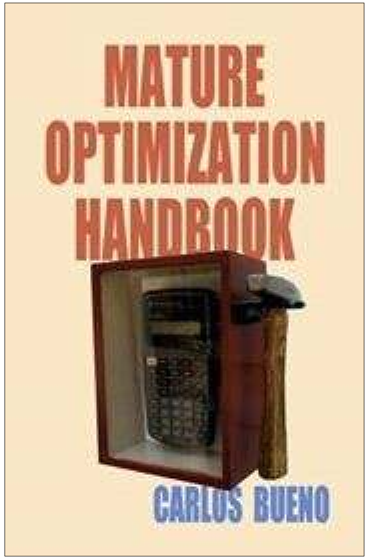


Structured Programming with Goto Statements, 1971

Mature Optimization Handbook (Carlos Bueno @ Facebook, 2013)

Performance optimization is, or should be a cost/benefit decision

In my experience, it makes most sense on mature systems whose architectures have settled down. New code is almost by definition slow code, but it's also likely to be ripped out and replaced as a young program slouches towards beta-test. Unless your optimizations are going to stick around long enough to pay for the time you spend making them, plus the opportunity cost of not doing something else, it's a net loss



https://www.facebook.com/notes/facebook-engineering/the-mature-optimization-handbook/10151784131623920/

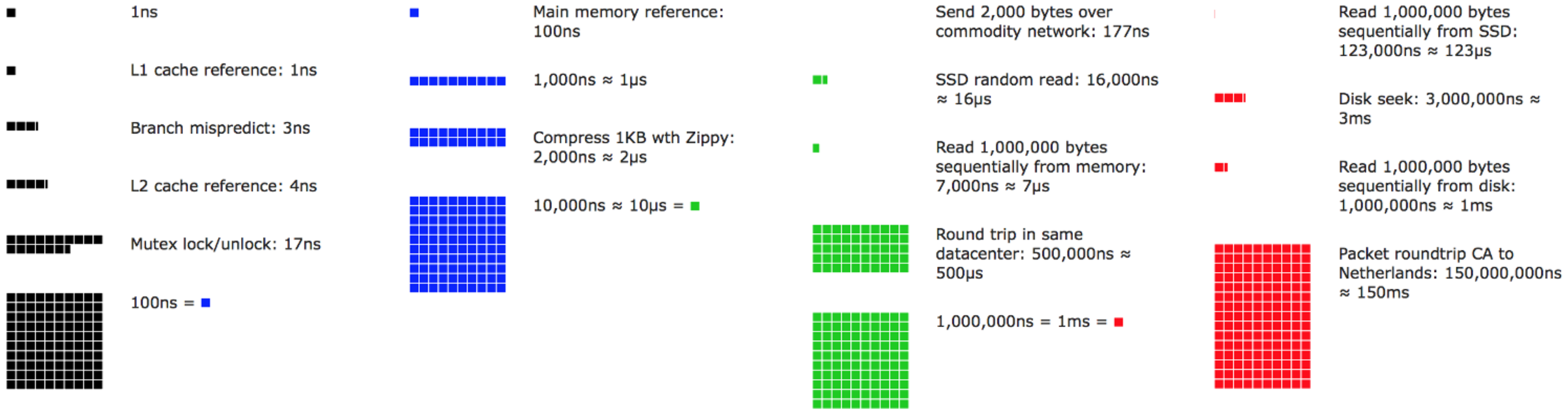
What's the ROI of saving *x* ms on a feature?

You don't know? My point exactly!

Find out before launching into a useless, risky optimization crusade

Latency Numbers Every Programmer Should Know

Actually, the numbers don't matter, the ratios do!



1ns

L1 cache reference: 1ns

Branch mispredict: 3ns

L2 cache reference: 4ns

Mutex lock/unlock: 17ns

100ns = 🟦

Main memory reference: 100ns

1,000ns ≈ 1µs

Compress 1KB wth Zippy: 2,000ns ≈ 2µs

10,000ns ≈ 10µs = 🟩

Send 2,000 bytes over commodity network: 177ns

SSD random read: 16,000ns ≈ 16µs

Read 1,000,000 bytes sequentially from memory: 7,000ns ≈ 7µs

Round trip in same datacenter: 500,000ns ≈ 500µs

1,000,000ns = 1ms = 🟥

Read 1,000,000 bytes sequentially from SSD: 123,000ns ≈ 123µs

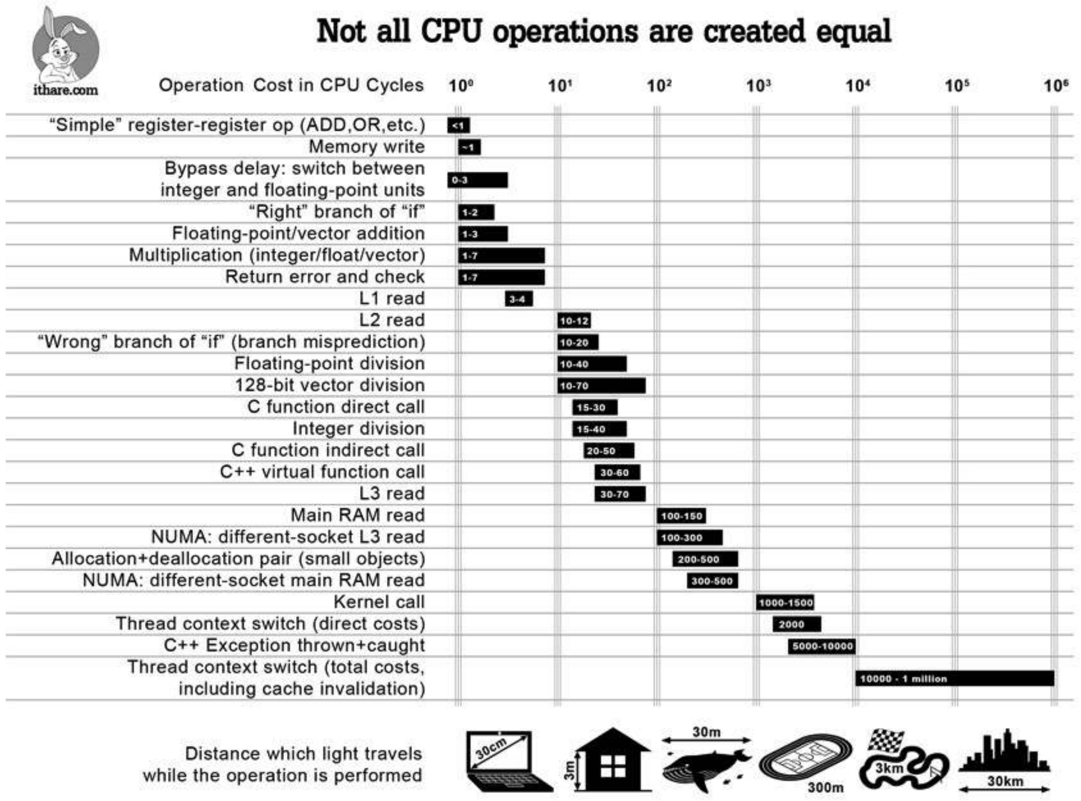
Disk seek: 3,000,000ns ≈ 3ms

Read 1,000,000 bytes sequentially from disk: 1,000,000ns ≈ 1ms

Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms

https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

Numbers only 0.1% of Programmers Should Know



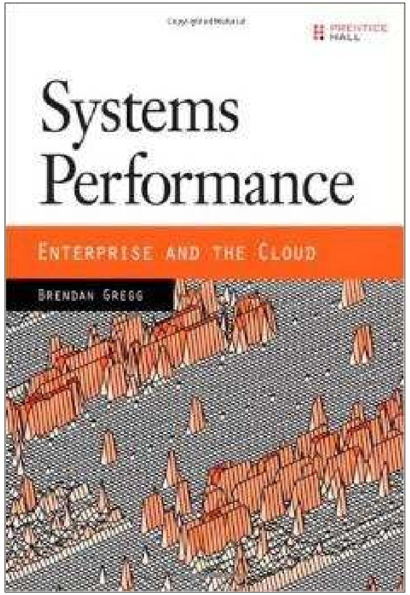
Operation	Cost in CPU Cycles
"Simple" register-register op (ADD,OR,etc.)	<1
Memory write	~1
Bypass delay: switch between integer and floating-point units	0-3
"Right" branch of "if"	1-2
Floating-point/vector addition	1-3
Multiplication (integer/float/vector)	1-7
Return error and check	1-7
L1 read	3-4
L2 read	10-12
"Wrong" branch of "if" (branch misprediction)	10-20
Floating-point division	10-40
128-bit vector division	10-70
C function direct call	15-30
Integer division	15-40
C function indirect call	20-50
C++ virtual function call	30-60
L3 read	30-70
Main RAM read	100-150
NUMA: different-socket L3 read	100-300
Allocation+deallocation pair (small objects)	200-500
NUMA: different-socket main RAM read	300-500
Kernel call	1000-1500
Thread context switch (direct costs)	2000
C++ Exception thrown+caught	5000-10000
Thread context switch (total costs, including cache invalidation)	10000 - 1 million

- Are you writing critical embedded code?
- Or compilers?
- Or HFT code?
- No? Keep out, then

http://ithare.com/infographics-operation-costs-in-cpu-clock-cycles /

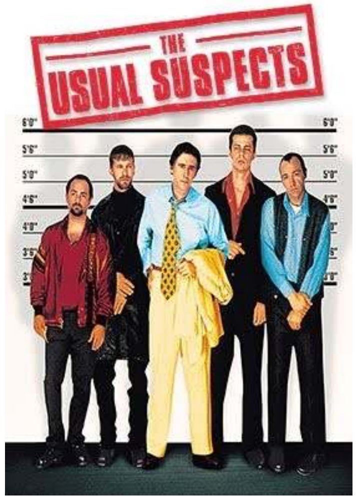
What I think really matters

- Make sure that critical product features are "fast enough"
- Don't wait after product launch to find out that they they're not.
- Performance monitoring is part on the dev process
  - Unit testing for raw performance
  - Load testing for throughput
  - Profiling: gprof, Callgrind, Dtrace, APM tools, etc.
- Most of the issues will be design or configuration, so start there instead of chasing mythical deep-sea monsters
- And yes, read <u>everything</u> Brendan Gregg writes.



Silent but Deadly (in no particular order)

1. Strings (StringBuilder, dammit)
2. Naive Memory / Thread Allocation (pooling, dammit)
3. 1+2 inside a loop
4. Database contention, misbehaving connection pools
5. In-memory cache contention (memcached, etc.)
6. Blocking calls and locks in general
7. Garbage Collection in “Workstation Mode”
8. Stupid SQL, too few or too many indexes, naive ORMs
9. Too many A/B tests or logs “that someone needed”
10. Parsing data (XML, JSON, etc) without a parser “because it’s simpler”
11. ...



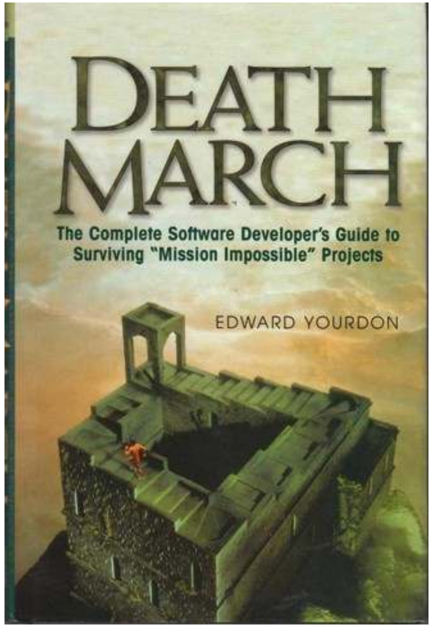
The Bleeding Edge

you’re gonna bleed, oh yes you are

Death march projects (Yourdon, 1996)

What really contributes to the death march nature of such projects […] is the attempt to use bleeding-edge technology for an industrial-strength application. Even if the technology is basically usable, it often does not scale up well for large-scale usage; and nobody knows how to exploit its strengths and avoid its weaknesses; and the vendors don't know how to support it properly; and on and on…

Is it any wonder that things degenerate into a death march project, with everyone working late nights and long weekends in order to coax the experimental new technology into some semblance of working order?


DEATH MARCH
The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects
EDWARD YOURDON

Choose boring technology (McKinley, 2015)

What counts as boring? That's a little tricky. "Boring" should not be conflated with "bad" […] There are many choices of technology that are boring and good, or at least good enough. MySQL is boring. Postgres is boring. PHP is boring. Python is boring. Memcached is boring. Squid is boring. Cron is boring.

The nice thing about boringness (so constrained) is that the capabilities of these things are well understood. But more importantly, their failure modes are well understood.

New technology choices might be purely additive (for example: "we don't have caching yet, so let's add memcached"). But they might also overlap or replace things you are already using. If that's the case, you should set clear expectations about migrating old functionality to the new system.

http://mcfunley.com/choose-boring-technology

Hype Driven Development (Kirejczyk, 2017)

Software development teams often make decisions about software architecture or technological stack based on inaccurate opinions, social media, and in general on what is considered to be "hot", rather than solid research and any serious consideration of expected impact on their projects. I call this trend Hype Driven Development, perceive it harmful and advocate for a more professional approach I call "Solid Software Engineering".

https://blog.daftcode.pl/hype-driven-development-3469fc2e9b22

Horror Stories Are Just… Stories

- Every single piece of technology has its own horror stories
- They’re fun to read and may teach you a thing or two
- However, they’re not always relevant to your platform
  - Different version
  - Different workload
  - Different scale
  - etc.
- So do not base any technology choice on horror stories

- ESBs and other shit Enterprise middleware. Just say no.

- Remember: it’s easier to blame tech than to expose your own mistakes

And please don't be one of these

- Trolls: « *Java is for pussies. Real men use C++* »
- Lunatics: « *Erlang is the future* ». Also works with Scala, Haskell, Clojure, etc.
- Old fart: « *SQL Server has always worked for us* » (and they buy us lunch)
- PhD egghead: « *we can reduce prediction latency by tweaking the Ethernet driver* »

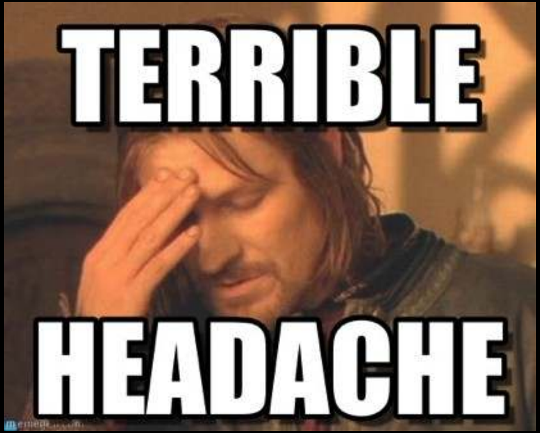
- And the worst of all, fanboys & hipsters:
  « *guys, HackyLib v0.1 has just been pushed to Github. It's totally awesome. Spotify, Netflix and Valve are already using it in production. Let's use it too!* »

Make your own choices, build your own platform.

Use common sense

1. Focus on business needs
2. Identify top challenges: time to market? UI? Perf? Security? Don't know?
3. List candidate technologies, expecting them to last at least a year (think 10x)
4. KPIs, benchmarks, PoC: educated guess is OK, random decision isn't!
5. Implement, deploy and monitor
6. Anything on fire?
   - Can it be fixed by refactoring or optimization?
   - If not (are you really sure?), can it be fixed with new technology?
     - Yes: you need a new building block in your stack, GOTO 2.
     - No: WTF? Are you scared? Man up! Not moving = death
   - If there is absolutely no other way, add servers…
     but it won't work forever!

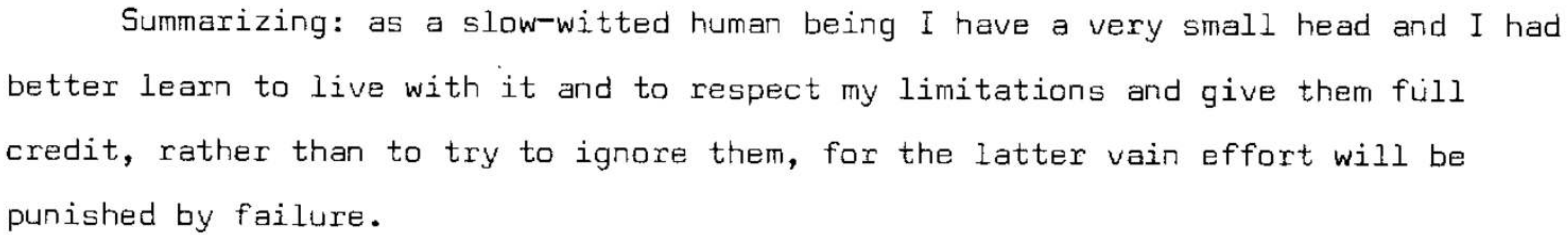
Summing things up



- Your project will die unless it finds product-market fit fast enough
- Unless you’re GAFA / FANG / NATU, you have the same ideas, tech and algos as everyone else: this isn’t how you’ll win the fight

- Turn ideas into MVPs as fast and frugally as possible
  - Lean & Agile methods
  - Flexible design, well-understood technology choices
  - Best development tools possible
  - Solid data stack to collect, store, process and visualize
- Keep a critical eye out for new tech, but think twice (or more) before adding it to your stack
- When C-level pushes the button, be ready to scale

Notes on Structured Programming (Dijkstra, 1969)



Summarizing: as a slow-witted human being I have a very small head and I had better learn to live with it and to respect my limitations and give them full credit, rather than to try to ignore them, for the latter vain effort will be punished by failure.

Thank you!

julien@julien.org
@julsimon

*“All that is gold does not glitter,
Not all those who wander are lost;
The old that is strong does not wither,
Deep roots are not reached by the frost”*