TDD, CI, and DevOps

Notes & Assumptions

- The organization is primarily focused on products development and large-scale projects, not small/ad-hoc projects. Which makes the long-term benefits and ROI clearly outweigh the initial overheads of making a transition (ex., training, adapting to new systems and processes, absorbing initial reduction in output in exchange for subsequently much faster and increased output, flexibility, and quality)
- Benefits are stated in no particular order (not by importance or significance for ex.)
- References

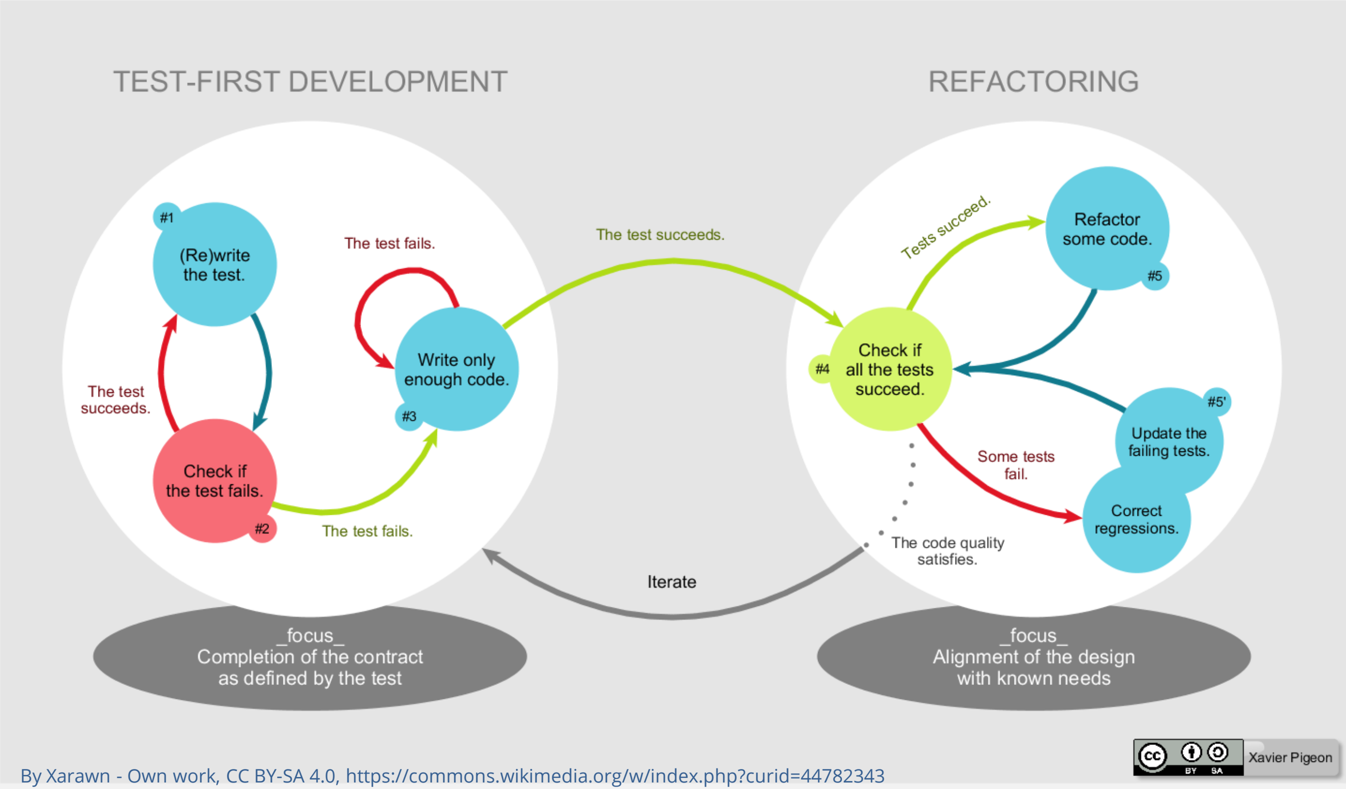
Contents Summary

- CI, TDD, DevOps definitions and processes
- Benefits
- Figures and statistics
- Making a shift (and considerations for developers)

Definitions and Processes - TDD

– Test-Driven Development

- “a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved to pass the new tests, only. This is opposed to software development that allows software to be added that is not proven to meet requirements”
  - Wikipedia

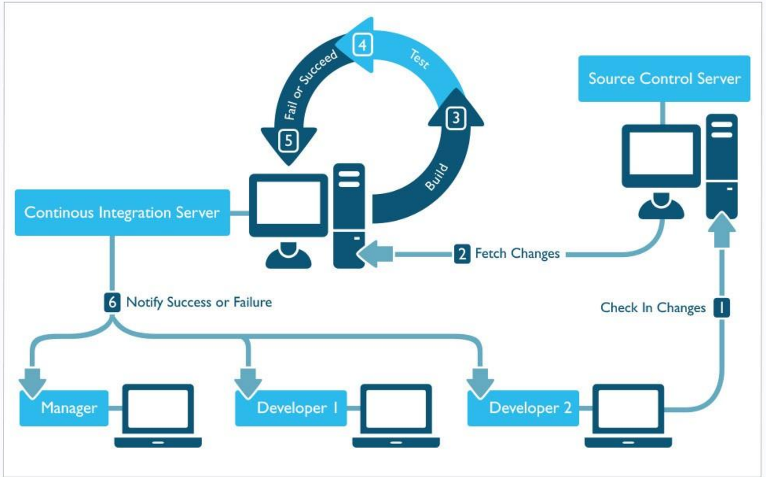


By Xarawn - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44782343

Definitions and Processes - CI

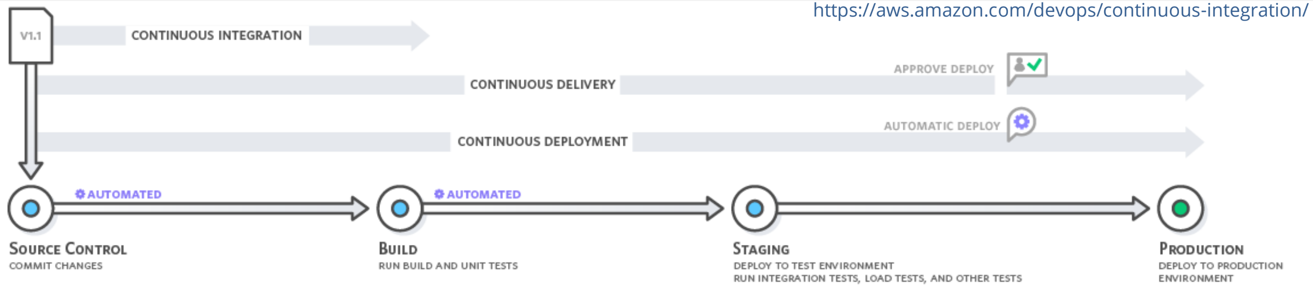
– Continuous Integration

- “is a development practice where developers integrate code into a shared repository frequently, preferably several times a day. Each integration can then be verified by an automated build and automated tests. While automated testing is not strictly part of CI it is typically implied”
  - Codeship
- “CI was intended to be used in combination with automated unit tests written through the practices of test-driven development”
  - Wikipedia



insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html

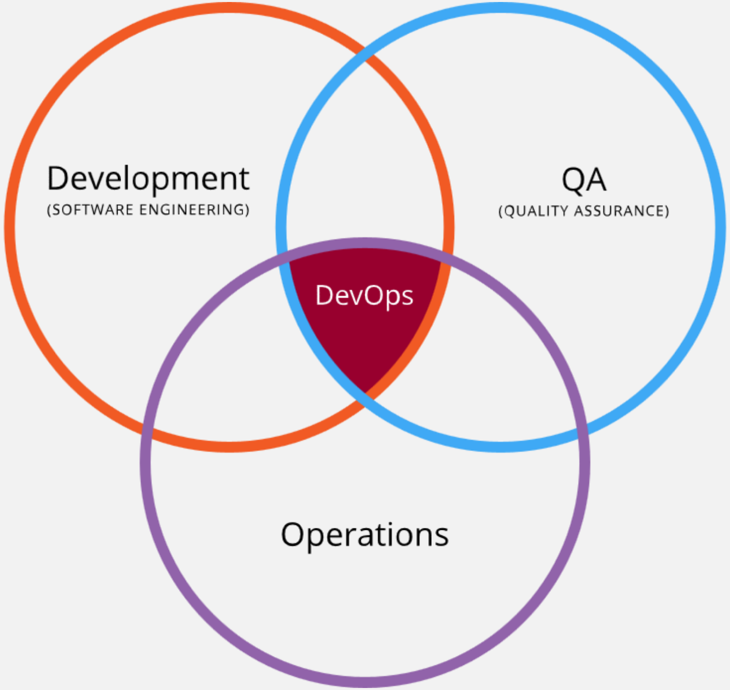
https://aws.amazon.com/devops/continuous-integration/

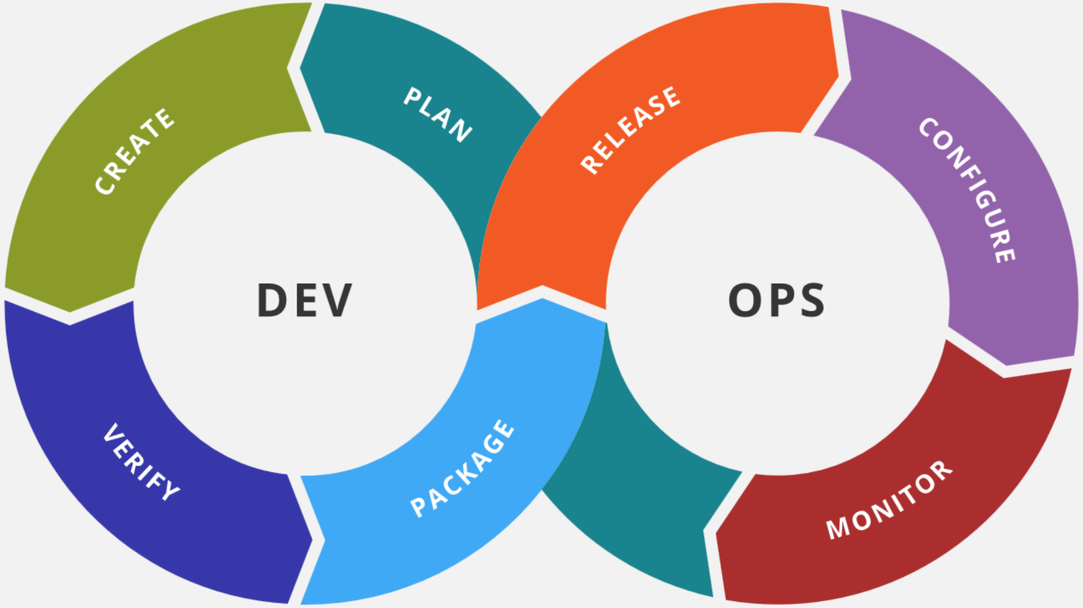


Definitions and Processes - DevOps

- DevOps

en.wikipedia.org/wiki/DevOps#/media/File:Devops.svg





en.wikipedia.org/wiki/DevOps#/media/File:Devops-toolchain.svg

Definitions and Processes - DevOps

– DevOps

- “DevOps (a clipped compound of "software DEVelopment" and "information technology OPerationS") is a term used to refer to a set of practices that emphasize the collaboration and communication of both software developers and information technology (IT) professionals while automating the process of software delivery and infrastructure changes. It aims at establishing a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably”
  - Wikipedia

- “DevOps aims to maximize the predictability, efficiency, security, and maintainability of operational processes”
  - Wikipedia

- “DevOps is the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support”
  - Theagileadmin

- Implementing DevOps
  - Systems Thinking
  - Amplifying Feedback Loops
  - Continual experimentation and learning

Benefits

– TDD

- Reduced debugging effort
- leads to more modularized, flexible, and extensible code (better design)
- High automated test coverage for code paths
- Increased confidence in software quality (requirements, and code works/runs)
- Reduction in defect rates
- Improve developer productivity (long-term, critical analytical thinking and design before writing code)
- Reduced maintenance and customer service cost

– CI

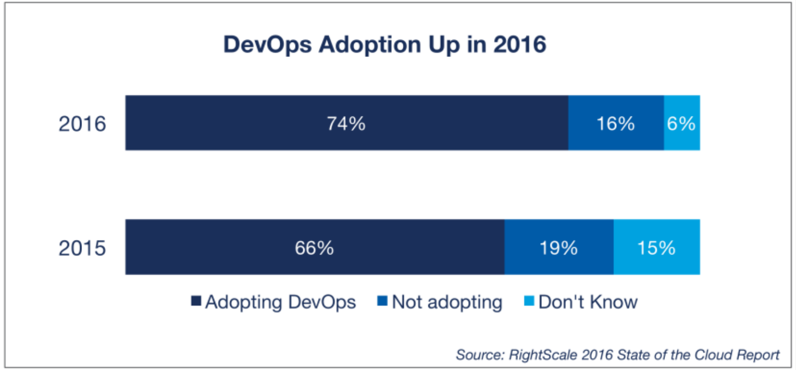
- Early detection of bugs and code errors, fixing is easier and cheaper
- Risk reduction (early discovery)
- Faster iterations
- Facilitates faster feedback and communication
- Enhanced (more automated) deployment process, which reduces overheads
- Reduced integration time and effort
- Reduces manual testing efforts

– DevOps

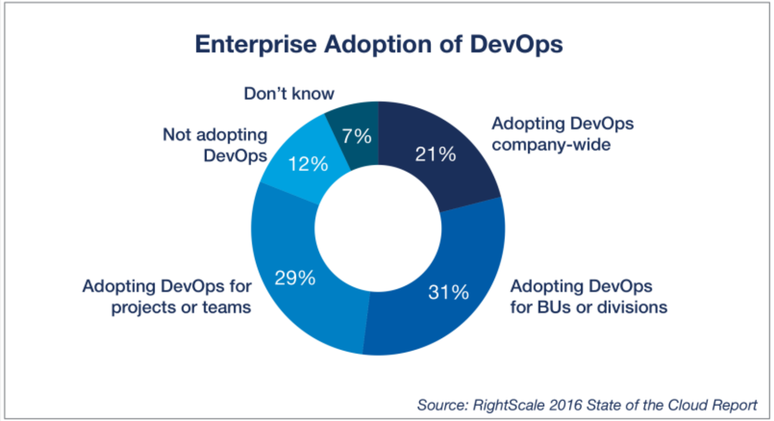
- Faster time to market
- Improved customer satisfaction
- Better quality, less failure rate, better reliability
- Improved productivity and efficiency
- Faster resolution of problems
- Higher employee engagement
- Improved communication and collaboration
- More time to build/innovate (rather than fix/maintain)

Figures and Statistics

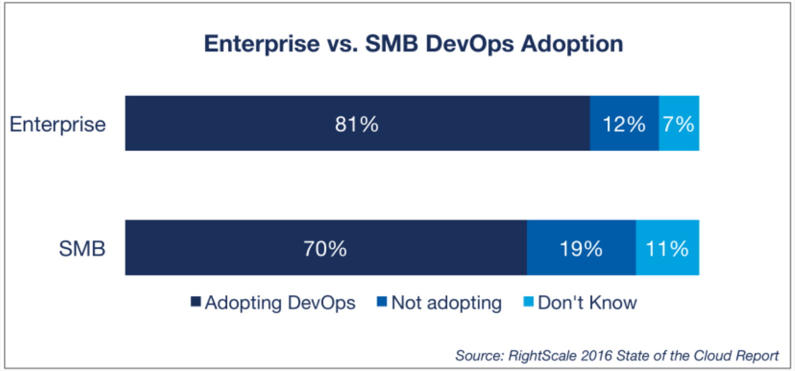
- Traditional Ops are 41% more time-consuming overall
- Traditional Ops spends an average of 7.2 hours weekly on communication
- Traditional Ops spends 21% more time putting out fires
- DevOps spends 33% more time on infrastructure improvements
- DevOps spends 60% less time handling support cases
- 63% experience improvement in the quality of their software deployments
- 63% release new software more frequently
- 55% notice improved cooperation and collaboration
- 38% report a higher quality of code production
(upguard.com/blog/devops-success-stats)



Source: RightScale 2016 State of the Cloud Report

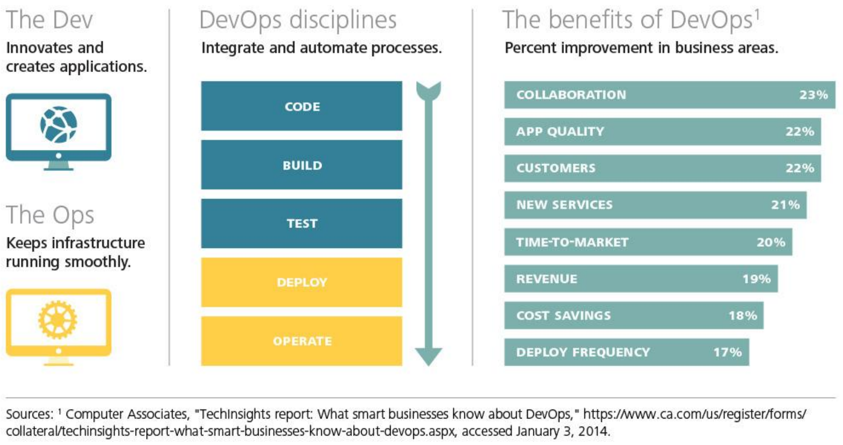


Source: RightScale 2016 State of the Cloud Report



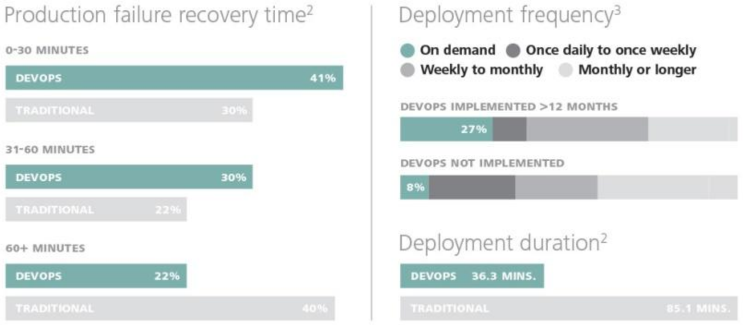
Source: RightScale 2016 State of the Cloud Report

Figures and Statistics



Sources: [1] Computer Associates, "TechInsights report: What smart businesses know about DevOps," https://www.ca.com/us/register/forms/collateral/techinsights-report-what-smart-businesses-know-about-devops.aspx, accessed January 3, 2014.

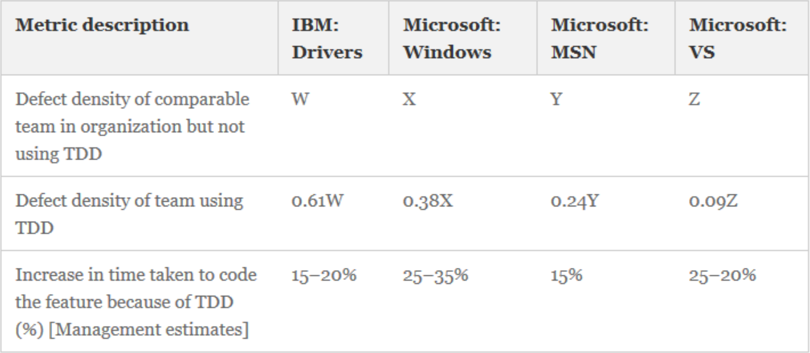
dupress.deloitte.com/dup-us-en/focus/tech-trends/2014/2014-tech-trends-real-time-devops.html



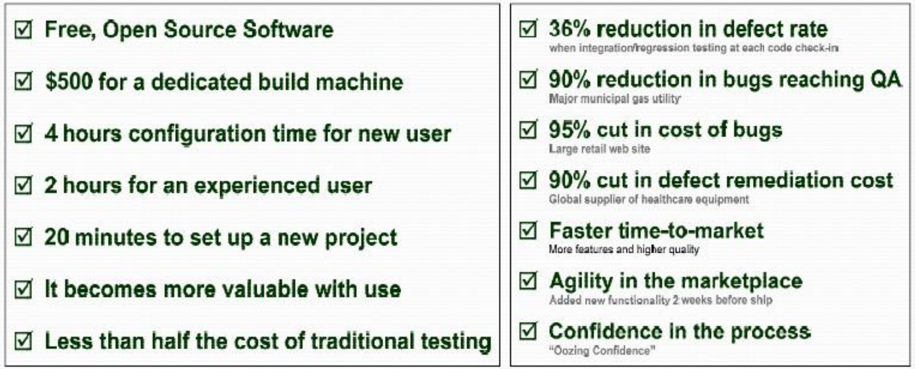
[2] ZeroTurnaround, "Rebel Labs IT ops & DevOps productivity report 2013," April 9, 2013, http://zeroturnaround.com/rebellabs/rebel-labs-release-it-ops-devops-productivity-report-2013, accessed January 6, 2014. [3] Puppet Labs, "2013 DevOps report,"March 2013, http://info.puppetlabs.com/2013-state-of-devops-report.html, accessed January 6, 2014.

dupress.deloitte.com/dup-us-en/focus/tech-trends/2014/2014-tech-trends-real-time-devops.html

link.springer.com/article/10.1007%2Fs10664-008-9062-z



Metric description	IBM: Drivers	Microsoft: Windows	Microsoft: MSN	Microsoft: VS
Defect density of comparable team in organization but not using TDD	W	X	Y	Z
Defect density of team using TDD	0.61W	0.38X	0.24Y	0.09Z
Increase in time taken to code the feature because of TDD (%) [Management estimates]	15–20%	25–35%	15%	25–20%



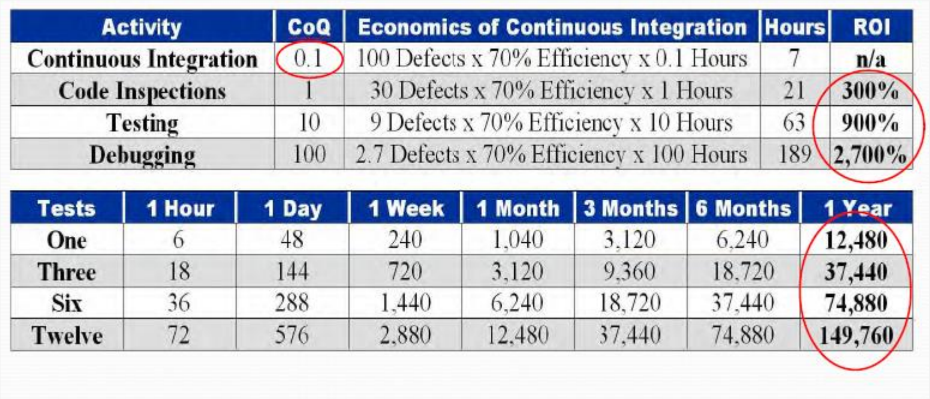
- ☑ Free, Open Source Software
- ☑ $500 for a dedicated build machine
- ☑ 4 hours configuration time for new user
- ☑ 2 hours for an experienced user
- ☑ 20 minutes to set up a new project
- ☑ It becomes more valuable with use
- ☑ Less than half the cost of traditional testing

- ☑ 36% reduction in defect rate
  when integration/regression testing at each code check-in
- ☑ 90% reduction in bugs reaching QA
  Major municipal gas utility
- ☑ 95% cut in cost of bugs
  Large retail web site
- ☑ 90% cut in defect remediation cost
  Global supplier of healthcare equipment
- ☑ Faster time-to-market
  More features and higher quality
- ☑ Agility in the marketplace
  Added new functionality 2 weeks before ship
- ☑ Confidence in the process
  "Oozing Confidence"

Grant, T. (2005). Continuous integration using cruise control. *Northern Virginia Java Users Group (Novajug), Reston, Virginia, USA.*
Fredrick, J. (2008). Accelerate software delivery with continuous integration and testing. *Japanese Symposium on Software Testing, Tokyo, Japan.*

Figures and Statistics

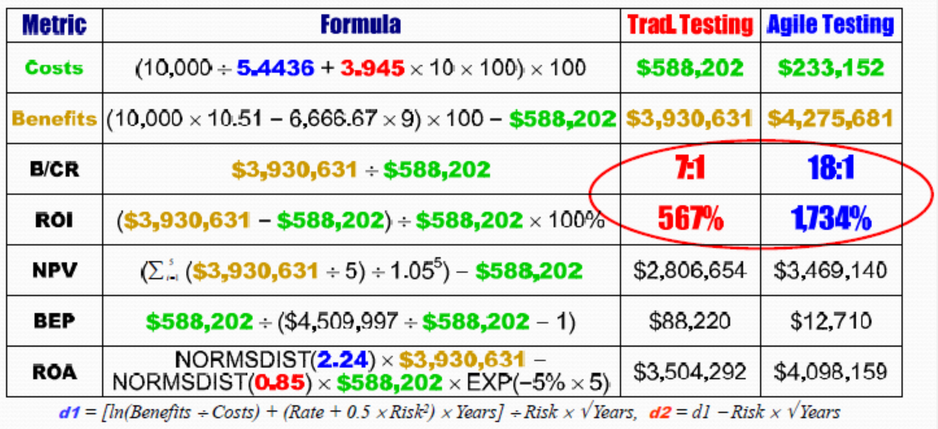
Abdulrahman Assabri - http://abdusabri.com



Activity	CoQ	Economics of Continuous Integration	Hours	ROI
Continuous Integration	0.1	100 Defects x 70% Efficiency x 0.1 Hours	7	n/a
Code Inspections	1	30 Defects x 70% Efficiency x 1 Hours	21	300%
Testing	10	9 Defects x 70% Efficiency x 10 Hours	63	900%
Debugging	100	2.7 Defects x 70% Efficiency x 100 Hours	189	2,700%

Tests	1 Hour	1 Day	1 Week	1 Month	3 Months	6 Months	1 Year
One	6	48	240	1,040	3,120	6,240	12,480
Three	18	144	720	3,120	9,360	18,720	37,440
Six	36	288	1,440	6,240	18,720	37,440	74,880
Twelve	72	576	2,880	12,480	37,440	74,880	149,760

Rico, D. F. (2012). *The Cost of Quality (CoQ) for Agile vs. Traditional Project Management*. Fairfax, VA: Gantthead.Com.

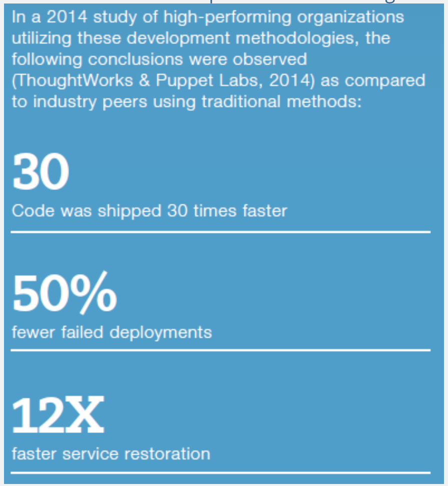


Metric	Formula	Trad. Testing	Agile Testing
Costs	(10,000 ÷ 5.4436 + 3.945 × 10 × 100) × 100	$588,202	$233,152
Benefits	(10,000 × 10.51 − 6,666.67 × 9) × 100 − $588,202	$3,930,631	$4,275,681
B/CR	$3,930,631 ÷ $588,202	7:1	18:1
ROI	($3,930,631 − $588,202) ÷ $588,202 × 100%	567%	1,734%
NPV	$(\sum_{i=1}^{5} (\$3{,}930{,}631 \div 5) \div 1.05^{5}) - \$588{,}202$	$2,806,654	$3,469,140
BEP	$588,202 ÷ ($4,509,997 ÷ $588,202 − 1)	$88,220	$12,710
ROA	NORMSDIST(2.24) × $3,930,631 − NORMSDIST(0.85) × $588,202 × EXP(−5% × 5)	$3,504,292	$4,098,159

$d1 = [\ln(Benefits \div Costs) + (Rate + 0.5 \times Risk^2) \times Years] \div Risk \times \sqrt{Years}$, $d2 = d1 - Risk \times \sqrt{Years}$

Rico, D. F., Sayani, H. H., & Sone, S. (2009). *The business value of agile software methods: Maximizing ROI with just-in-time processes and documentation*. Ft. Lauderdale, FL: J. Ross Publishing.

BETSOL-2015-White-Paper-Continuous-Integration



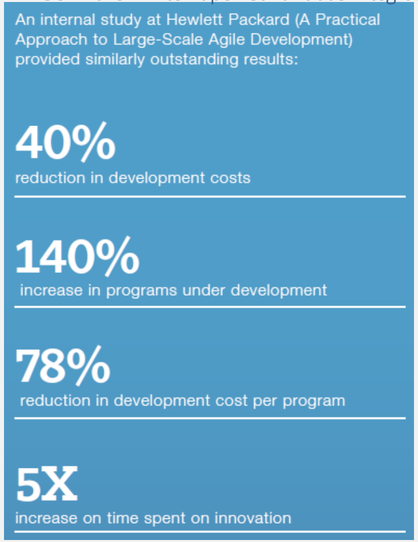
In a 2014 study of high-performing organizations utilizing these development methodologies, the following conclusions were observed (ThoughtWorks & Puppet Labs, 2014) as compared to industry peers using traditional methods:

30
Code was shipped 30 times faster

50%
fewer failed deployments

12X
faster service restoration

BETSOL-2015-White-Paper-Continuous-Integration



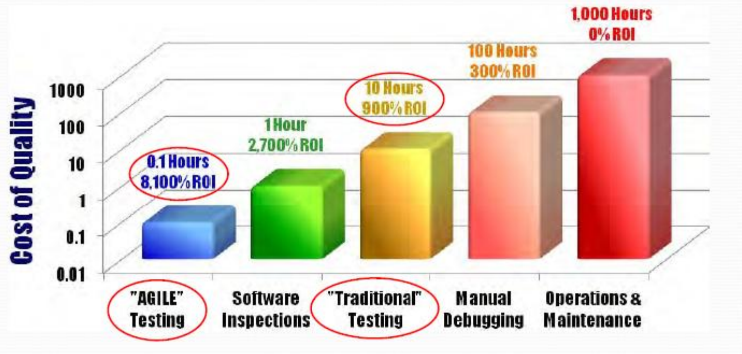
An internal study at Hewlett Packard (A Practical Approach to Large-Scale Agile Development) provided similarly outstanding results:

40%
reduction in development costs

140%
increase in programs under development

78%
reduction in development cost per program

5X
increase on time spent on innovation



Rico, D. F. (2012). *The Cost of Quality (CoQ) for Agile vs. Traditional Project Management*. Fairfax, VA: Gantthead.Com.

Figures and Statistics

- □ Hewlett-Packard is a major user of CI, CD, & DevOps
- □ 400 engineers developed 10 million LOC in 4 years
- □ Major gains in testing, deployment, & innovation

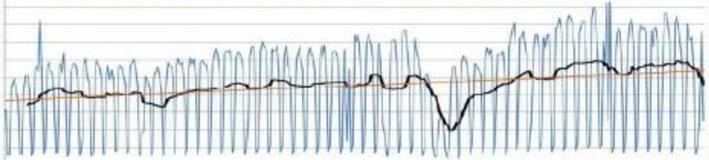
Type	Metric	Manual	DevOps	Major Gains
Cycle Time Improvements	Build Time	40 Hours	3 Hours	13 x
	No. Builds	1-2 per Day	10-15 per Day	8 x
	Feedback	1 per Day	100 per Day	100 x
	Regression Testing	240 Hours	24 Hours	10 x
Development Cost Effort Distribution	Integration	10%	2%	5 x
	Planning	20%	5%	4 x
	Porting	25%	15%	2 x
	Support	25%	5%	5 x
	Testing	15%	5%	3 x
	Innovation	5%	40%	8 x

Gruver, G., Young, M. & Fulghum, P. (2013). *A practical approach to large-scale agile development*. Upper Saddle River, NJ: Pearson Educat

Whittaker, J.

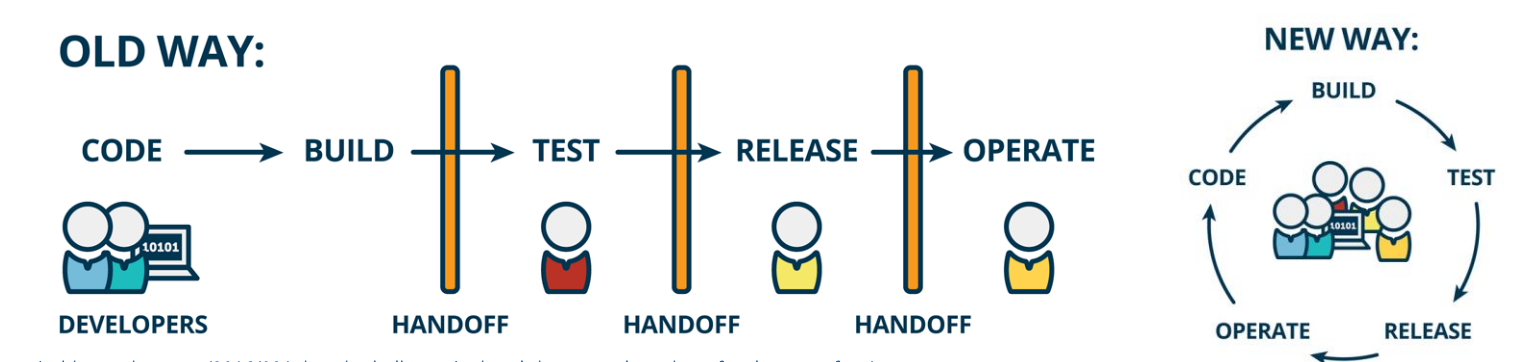
- 440 billion unique users run 37 trillion searches each year
- Single monolithic code tree with mixed language code
- Submissions at head – One branch – All from source
- 20+ code changes/minute – 50% code change/month
- 5,500+ submissions/day – 120 million tests per day
- 80,000 builds per day – 20 million builds per year
- Auto code inspections – For low defect density
- 10X programming productivity improvement
- **$150 million in annual labor savings** (ROI as a result)

- Software deployment every 11.6 seconds (as of 2011)
  - → 24,828 to 86,320 releases per Iteration
  - → 161,379 to 561,080 releases per Quarter
  - → 645,517 to 2,244,320 releases per Year
- Automatic, split-second roll-forward & backward
- 75-90% reduction in release-caused outages (0.001%)
- Millions of times faster (than traditional methods)
  - → 4,357,241 to 15,149,160 per traditional release
- Thousands of times faster (than manual agility)
  - → 161,379 to 561,080 per Scrum/SAFe release
- **Used agile methods long before U.S. government** (1999)



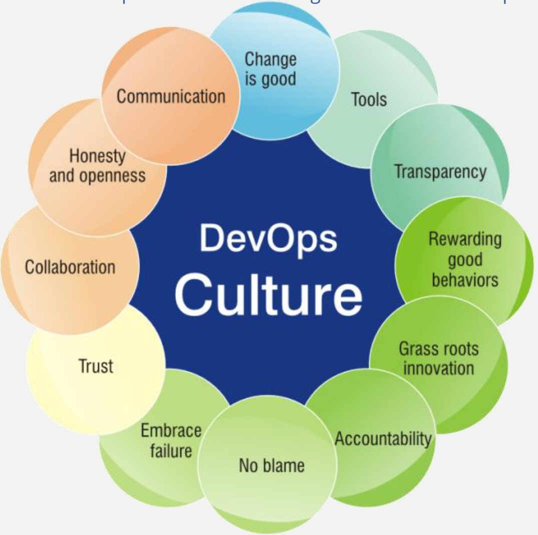
Atlas, A. (2009). Accidental adoption: The story of scrum at amazon.com. *Proceedings of the Agile 2009 Conference, Chicago, Illinois, USA, 135-140.*
Jenkins, J. (2011). Velocity culture at amazon.com. *Proceedings of the Velocity 2011 Conference, Santa Clara, California, USA.*
Elisha, S. (2013). Continuous deployment with amazon web services. *Proceedings of the AWS Summit 2013, Sydney, New South Wales, Australia.*

Making a shift



mindtheproduct.com/2016/02/what-the-hell-are-ci-cd-and-devops-a-cheatsheet-for-the-rest-of-us/

supinfo.com/articles/single/3652-what-is-devops



– Considerations for Developers

- Agile mindset, culture, principles
- Critical analytical thinking about requirements and design before coding
- More unit testing
- Smaller problems and pieces of code at a time (vs big features, chunks of code)
- More frequent code check-ins and integration
- More collaboration (specially, with QA, IT/Infrastructure)