CI/CD at scale
Lessons from LinkedIn and Mockito

QCon, San Francisco, 11-2017

- Szczepan Faber @mockitoguy
- Born in Poland (we like our zzz's)
  In US since 2015

- Codes professionally since 2002
- Creator of mockito.org in 2007
- Core dev of gradle.org 1.x and 2.x in 2011-2015
- Tech Lead at LinkedIn Dev Tools since 2015
  http://bit.do/li-tools

- Author on LinkedIn: http://bit.do/mockitoguy
- How to build great code review culture? http://bit.do/li-code-review

- Want to write great tests?
  My workshop at QCon on Thursday:
  http://bit.do/qcon-testing
- Want to innovate and push CD in Open Source?
  Join shipkit.org
  New project used by Mockito!

Imagine productivity

without the release overhead

Get ready to be excited about CD!

1. CD at LinkedIn (@LinkedInEng)

- linkedin.com adopted CD in 2015, shipping 3x day to 500M users.
- I'm an architect, tech lead, engineer working on development tools at LinkedIn since 2015.
- Kudos to great engineers working at LinkedIn, linkedin.com and the tools. I am a part of a team.

LinkedIn Engineering (3000+)

Foundation team (300+)

Development Tools

- Build tools, Gradle, CI
- Code review, IDEs
- and more!

2. CD in OSS Mockito (@MockitoJava)

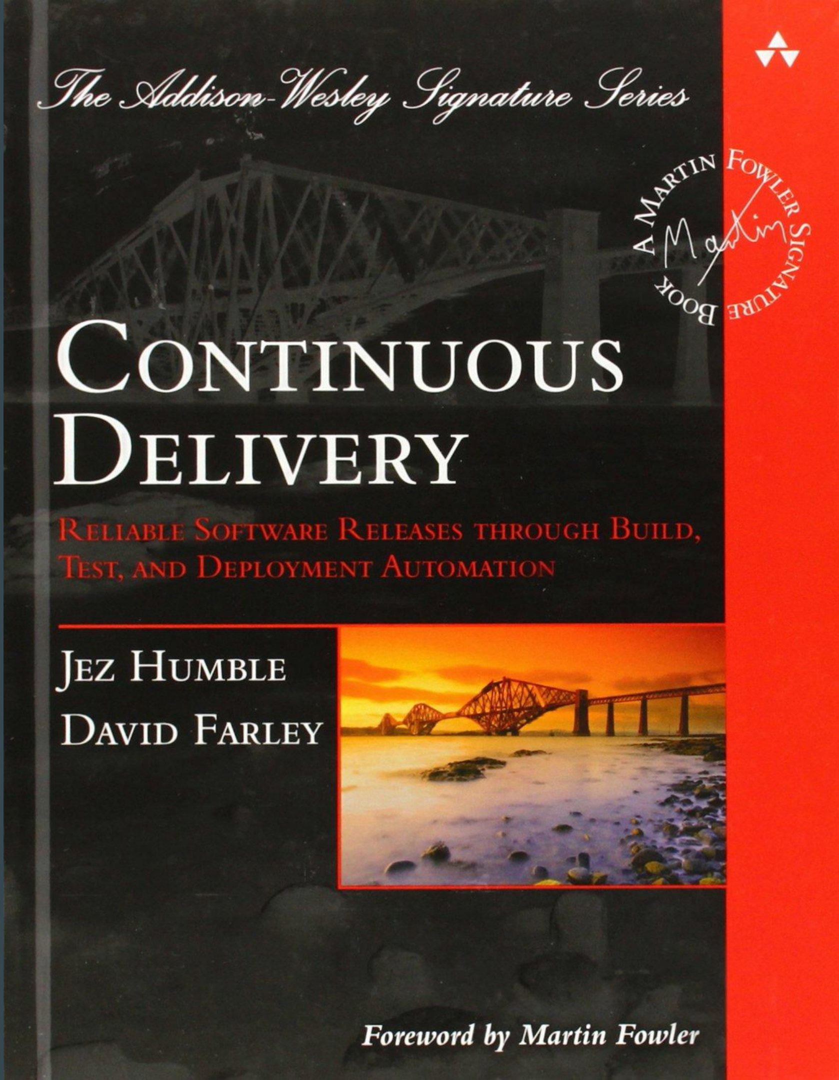
- I created mocking framework Mockito in 2007
- The team adopted CD in 2014
- We ship every pull request to production to estimated 2M users

CD at linkedin.com by the numbers

- Last week (5-11 Nov ‘17)
  - 1000 commits, 300 unique committers across 4 main codebases
  - Web app, API server app, Android app, iOS app
  - The stats exclude other codebases (libraries, downstream microservices)
  - LinkedIn does not use (nor wants or needs) mono-repo. We currently have 7000+ codebases.
- Last quarter (Q3 ‘17):
  - 300 pushes to production (web app + api server)
  - 21 mobile app pushes (android + iOS, excluding Beta)
- LinkedIn engineering is more than linkedin.com
  - Other products exercise CD as well but don’t have such big codebase or number of committers
  - linkedin.com (flagship) is very progressive and paves the way for other LinkedIn software products

Part 1.
CD is hard

What do we do if something is hard?


The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Continuous Delivery
Reliable Software Releases through Build, Test, and Deployment Automation
Jez Humble
David Farley
Foreword by Martin Fowler

Why CD in linkedin.com?

In 2015 we launched 3x3 project - 3 releases per day, 3 hour max time commit-to-production.
Our goals:

- reduce the lead time to make positive business impact
  - ship features to production faster
- increase engineering productivity and happiness
  - avoid release overhead
    - avoid wasted time on stabilizing (bugfixing) the release branch. Code should be always stable and ready to ship!
    - avoid wasted time on cherry-picking. Trunk based development.
  - avoid feature rush - last minute commit volume spike before the release
- improve quality
  - small incremental releases pose small risk and are easier to fix
  - avoid rollbacks and hotfixes. Instead we fix forward

The arguments apply to every software project!

3x3 @LI

- Why 3 times per day?
    - Because we want to iterate fast. Plus it helps with resilience because we can afford to miss a release
- Why 3 hours max time commit-to-production?
    - Because to ship 3 times a day, the commit-to-production pipeline needs speed
    - And it forces us to sort out our testing strategy (not enough time for manual or slow tests)
- In 2015 we completed mobile web and mobile apps
- In 2016 we completed desktop web
- Today we release linkedin.com several times a day
- LinkedIn Mobile apps are released every week, with 3 beta releases per week (iOS Beta - once a week)

CD lessons @LI

- Learning how to write great tests
  - what to test, how to test, how to write code that is easy to test
- Flaky test is worse than no test
  - detecting flaky tests automatically, overnight, using A/A testing
- Production grade tests and infrastructure
  - in the past, tests and build code were not considered equally important as production code
- Many existing tools, including OSS did not scale for CD
- Need for speed – testing in parallel and distributed
- Master branch always green
  - Running all validations before code is merged to master

Current 3x3 stats for linkedin.com

	Commits and unique committers in last week	Avg. time commit-to-shippable in last week	# of actual and target releases in Q3 2017
Web	317/115	150 min.	173/162
API	183/77	69 min.	123/162
iOS	164/69	165 min.	10/12
Android	241/72	72 min.	11/12
Combined	905*/333		

(*) in actuality, it is higher because commits to libraries are not included

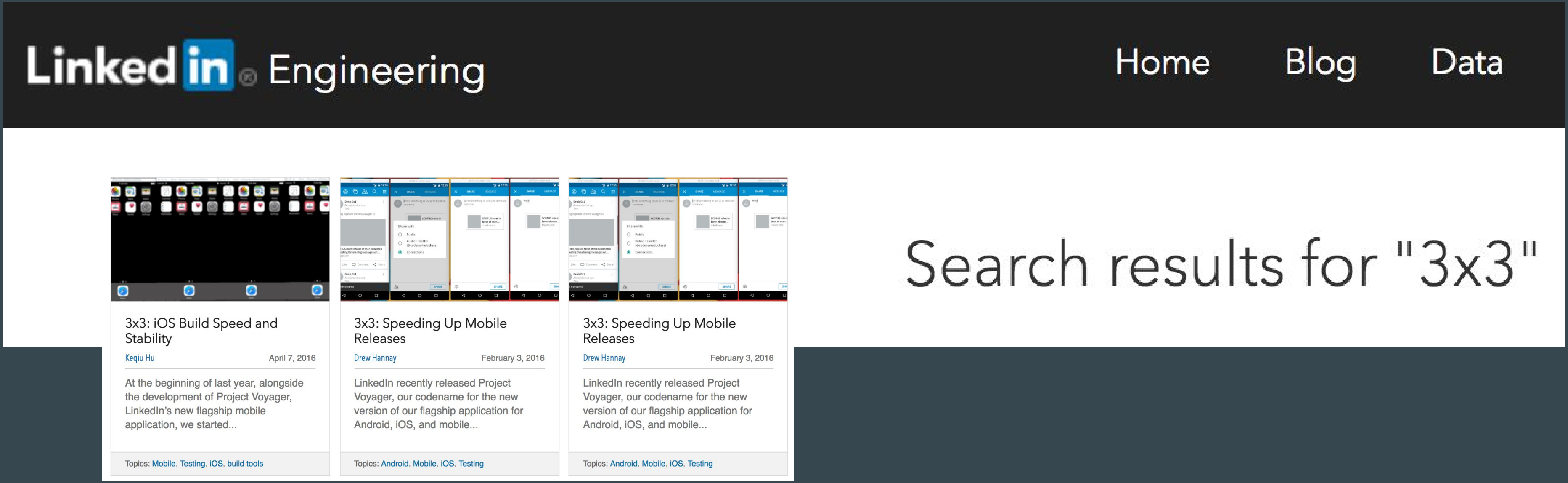
Opportunities @LI

- Minimize commit-to-shippable time
- Increase commit-to-shippable pipeline success rate (pipeline & tests stability)
- Hit the desired number of production releases
- Increase discipline of fixing flaky tests
- Avoid redundant work in the pipeline
- For Android: simulator service, speed
- For iOS: Beta channel, Swift compiler stability and speed
- For Web app: browser cache
- For API server: avoid redundant work (Gradle Distributed Cache)
  BTW. we use Play on Gradle for our API server: http://bit.do/play-on-gradle

And many more! We keep improving the system!!!

3x3 resources

LinkedIn Engineering Blog: http://bit.do/li-3x3



Search results for "3x3"

3x3: iOS Build Speed and Stability
Keqiu Hu — April 7, 2016
At the beginning of last year, alongside the development of Project Voyager, LinkedIn's new flagship mobile application, we started...
Topics: Mobile, Testing, iOS, build tools

3x3: Speeding Up Mobile Releases
Drew Hannay — February 3, 2016
LinkedIn recently released Project Voyager, our codename for the new version of our flagship application for Android, iOS, and mobile...
Topics: Android, Mobile, iOS, Testing

3x3: Speeding Up Mobile Releases
Drew Hannay — February 3, 2016
LinkedIn recently released Project Voyager, our codename for the new version of our flagship application for Android, iOS, and mobile...
Topics: Android, Mobile, iOS, Testing

Development Workflow @LI

CD requires disciplined development workflow.
How software is developed at LinkedIn? (all our software, not only linkedin.com)

- Multi-codebase architecture, 7000 codebases, 60% active
- Every codebase is governed by our "Multiproduct" framework
- Every Multiproduct has independent release cadence
- Every code change produces new version
- Trunk based development
- Mandatory code review (code owner must approve every change)
- Automation of commit-to-production pipeline

Every codebase is a Multiproduct @LI

- One engineering culture
- Every engineer can contribute to any codebase
- Why matters? Easier to introduce CD into one culture

Every change is a new version @LI

- And every new version can be shipped to production
- Why matters?
  - Makes it impossible to defer quality.
  - Clean code every day!

Trunk based development @LI

- No long-lived feature branches
- All changes on main branch, which is always stable and ready to ship
- Incremental code changes, hiding incomplete features
- Feature toggles, “branch by abstraction” pattern
  http://bit.do/branch-by-abstraction
- Why matters? Forces small, incremental changes. Avoids merge and cherry-picking overhead.

Mandatory code review @LI

- Somebody reads my code (and wants me to fix it)!
- Culture of feedback, learning and improving
  http://bit.do/li-code-review
- Why matters? Clean, elegant code makes it easier to iterate

Downstream dependency testing @LI

- Building code that depends on my code
- Strong signal in our CI pipeline
- Flaky tests are a problem to the entire ecosystem
- Why matters? avoiding regressions, catching integration problems early

Automation of commit-to-production @LI

- The code I write or review goes to production within hours
- I am responsible for the quality
- Why it matters? Requires state of the art automated testing
  - BTW. I run a Java testing workshop on Thursday at QCon http://bit.do/qcon-testing

CI/CD pipeline @LI

For all our software, not only components of linkedin.com

- Code change
- Code review (strong code ownership, owner must approve)
- Pre/post push validation (CI builds)
- Downstream dependency testing (test code that depend on me)
- New version ready!

Apps

- Staging
- Canary (deploy and test on single host)
  - Mobile uses Beta channel
- Ramp-up features (feature toggles)
  - Code push != feature push
- Remove feature toggle (dead code)

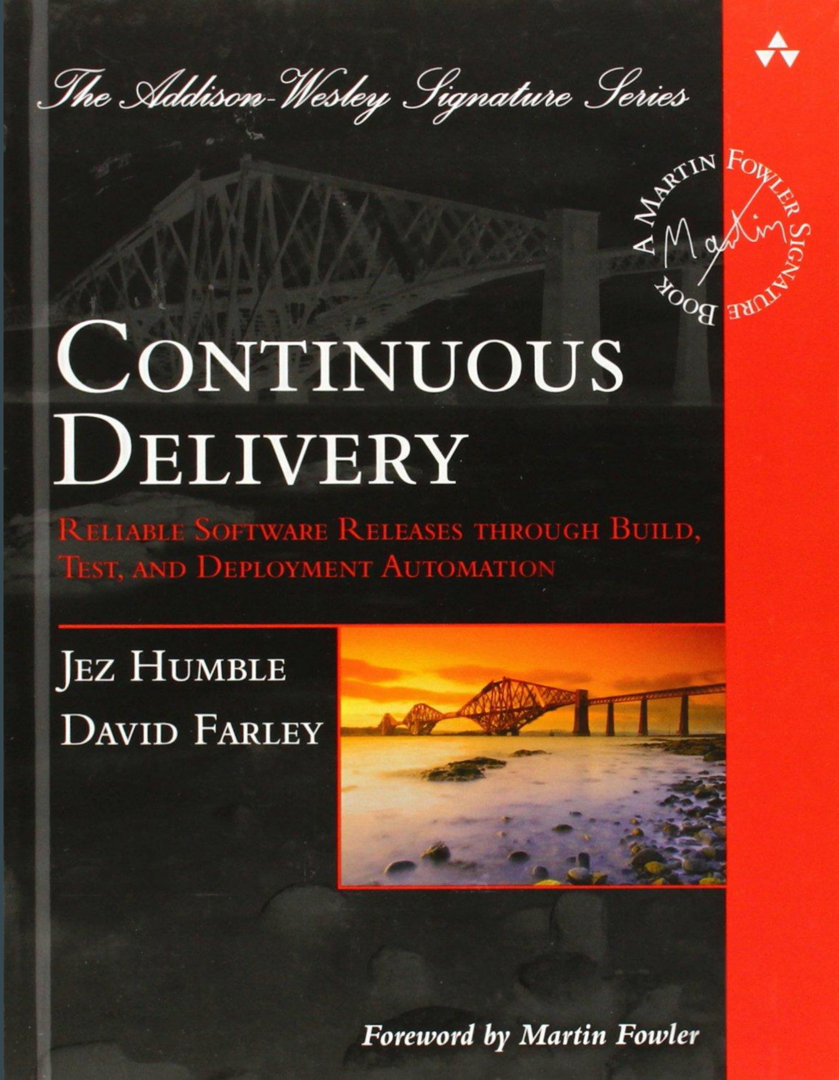
Libraries

- Consumers that use wildcard versions pick up new version in the next build.
- Consumers that use pinned version can be updated automatically using “Push my Upgrade” system.
- We can deprecate/end-of-life previous versions

Dev workflow for all software at LinkedIn. Ready for part 2 (OSS)?

Part 2.
CD. in OSS
Mockito

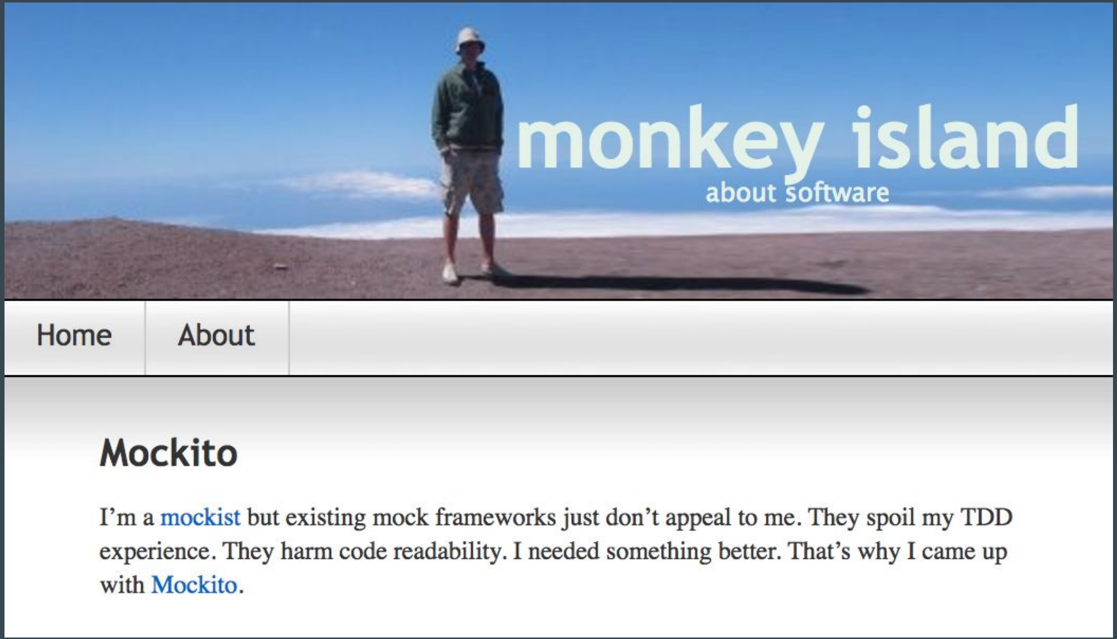
Powered by shipkit.org


The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Continuous Delivery
Reliable Software Releases through Build, Test, and Deployment Automation
Jez Humble
David Farley
Foreword by Martin Fowler

@mockitoguy

Open Source Mocking Framework for Java

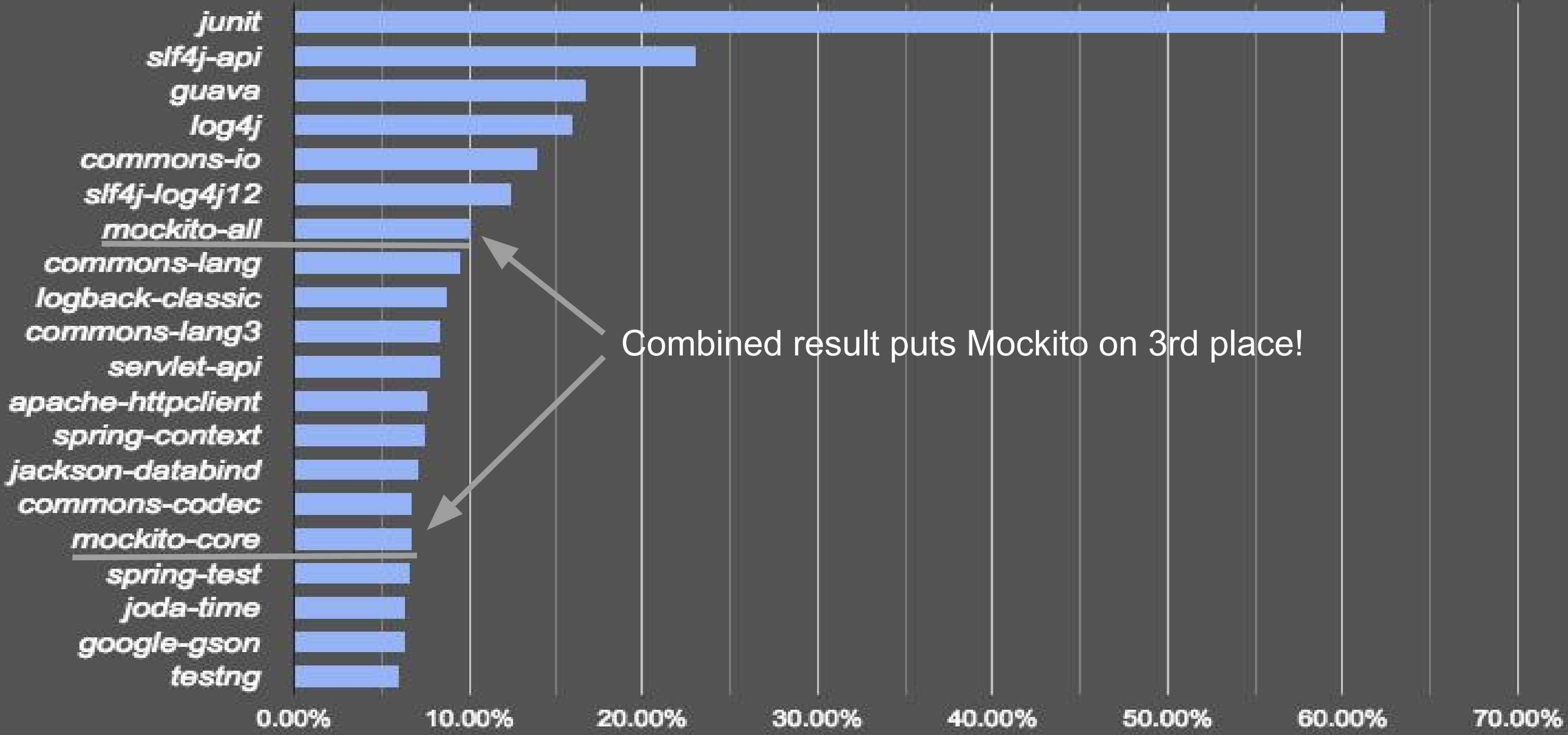
- Mockito started in 2007
- 1.0 in 2008
- Hit mainstream in 2010
- Mostly manual releases...



Mockito

I'm a mockist but existing mock frameworks just don't appeal to me. They spoil my TDD experience. They harm code readability. I needed something better. That's why I came up with Mockito.

The Top 20 Libraries Used by Github's Most Popular Java Projects

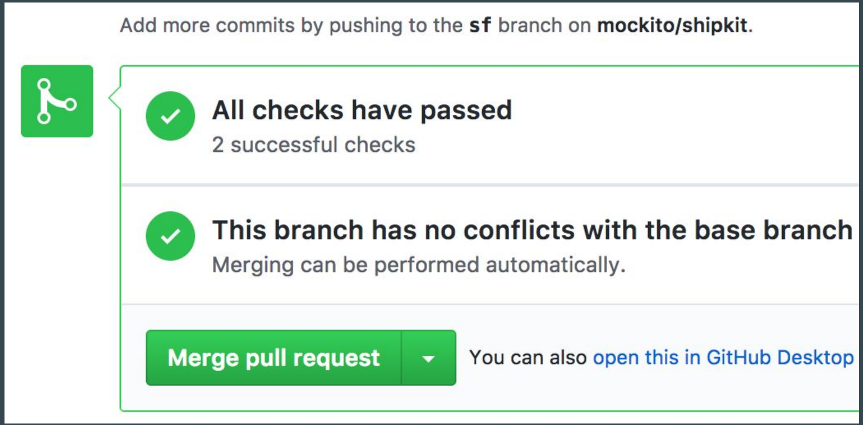


Combined result puts Mockito on 3rd place!

TAKIPI

2014 - Mockito adopts CD.

- Prevent release procrastination (dreading to write release notes...)
- Every merged pull request produces a new version and ships to public repo
- Scale: we estimate 2M users

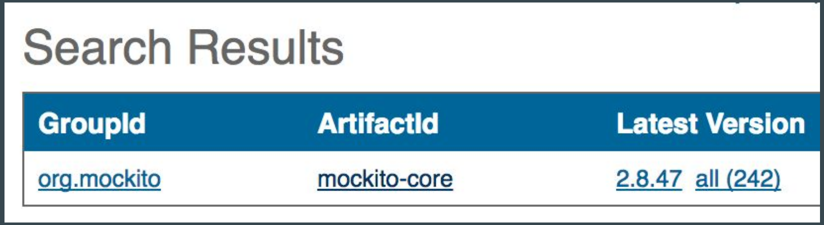


Add more commits by pushing to the **sf** branch on **mockito/shipkit**.

All checks have passed
2 successful checks

This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop



Search Results

GroupId	ArtifactId	Latest Version
org.mockito	mockito-core	2.8.47 all (242)

Benefits of CD in OSS

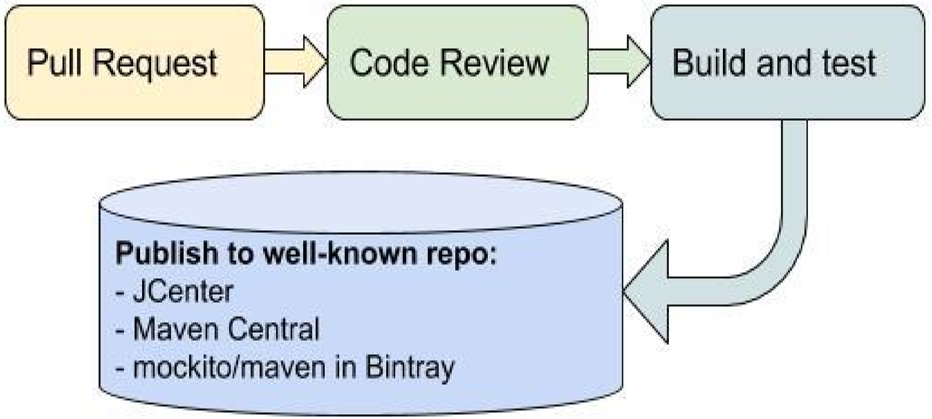
- Productivity – zero release overhead
- Happy users – get features faster
- Faster debugging – quickly identify bad version (MTTR)
- Sustainability – release & stay alive
- No waste - no unreleased code
- Quality – self-enforced craftsmanship of every change
- Thriving, engaged community – contributions are released quickly

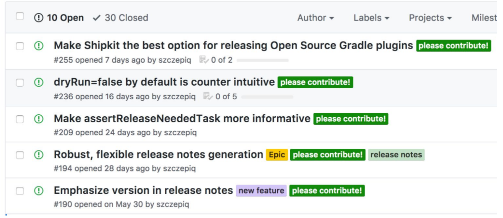
Community Feedback

- Quality anxiety
  - you ship every pull request to production, are you shipping every bug, too?
  - we ensure quality via immense battery of tests and rigorous code review
- What version of “mockito-core” to use?
  - currently 244 versions in Maven Central
  - use latest! We take compatibility VERY seriously. Sem ver!
- Dependency management cost
  - worry that any version upgrade may bring incompatibilities to the dependency graph
  - we get it. We strive to minimize Mockito dependencies. We understand that every dependency is a liability to our customers

Mockito releases By Shipkit

- Shipkit - toolkit for shipping it for Java libraries
- Passionate about release automation?
- http://shipkit.org




10 Open ✓ 30 Closed Author ▾ Labels ▾ Projects ▾ Milest

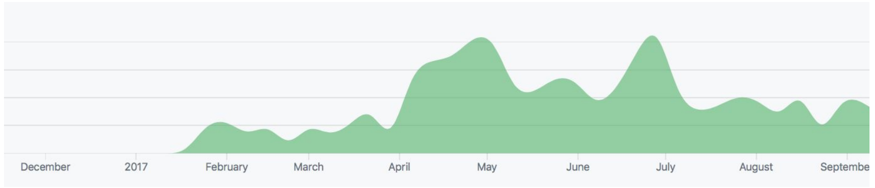
Make Shipkit the best option for releasing Open Source Gradle plugins please contribute!
#255 opened 7 days ago by szczepiq 0 of 2

dryRun=false by default is counter intuitive please contribute!
#236 opened 16 days ago by szczepiq 0 of 5

Make assertReleaseNeededTask more informative please contribute!
#209 opened 24 days ago by szczepiq

Robust, flexible release notes generation Epic please contribute! release notes
#194 opened 28 days ago by szczepiq

Emphasize version in release notes new feature please contribute!
#190 opened on May 30 by szczepiq



@mockitoguy

Ready for CD?

- Imagine how fast you can ship changes that can create positive business impact
- Imagine unhindered productivity without the release overhead
- Imagine higher quality because smaller, incremental releases are a smaller risk
- Imagine that every code change is excellent, with clean code and great tests
- Imagine how reliable the commit-to-production pipeline is if it is battle tested daily

- At linkedin.com we land 1000 commits per week and ship to production several times a day to 500M+ users.
- In the Open Source, Mockito library ships to production every pull request to estimated 2M users

Now it is your turn! Questions?