iOS Release Pipeline

A real world example of a distributed, in-house, release pipeline using Jenkins in an enterprise workspace.

generally speaking

It was 5 years ago…

tools

have changed

my memory

is not that good

a bit of context

and your imagination

this is the enterprise

multiple teams, departments, stakeholders

multiple environments

security reasons* (e.g. access to customer data)

feature branches

one for each user story

develop

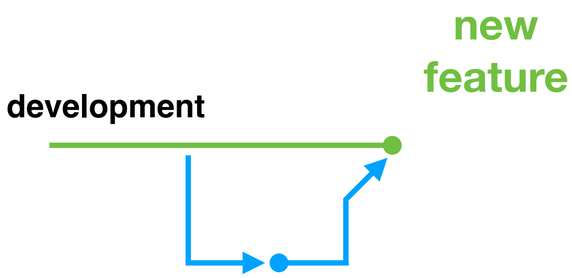
on a branch

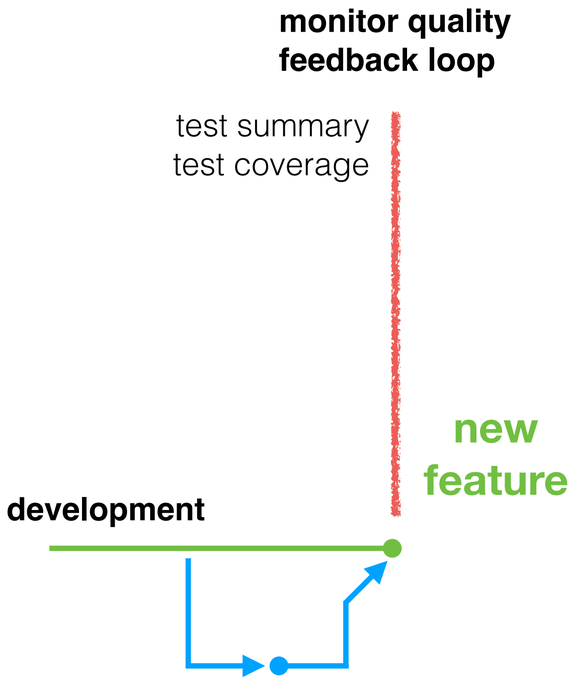
release

on master

so, how did it look?

an overview


new
feature
development


monitor quality
feedback loop
test summary
test coverage
new
feature
development

feedback loop

must be sort

did we break the build?

merges do that

monitor quality

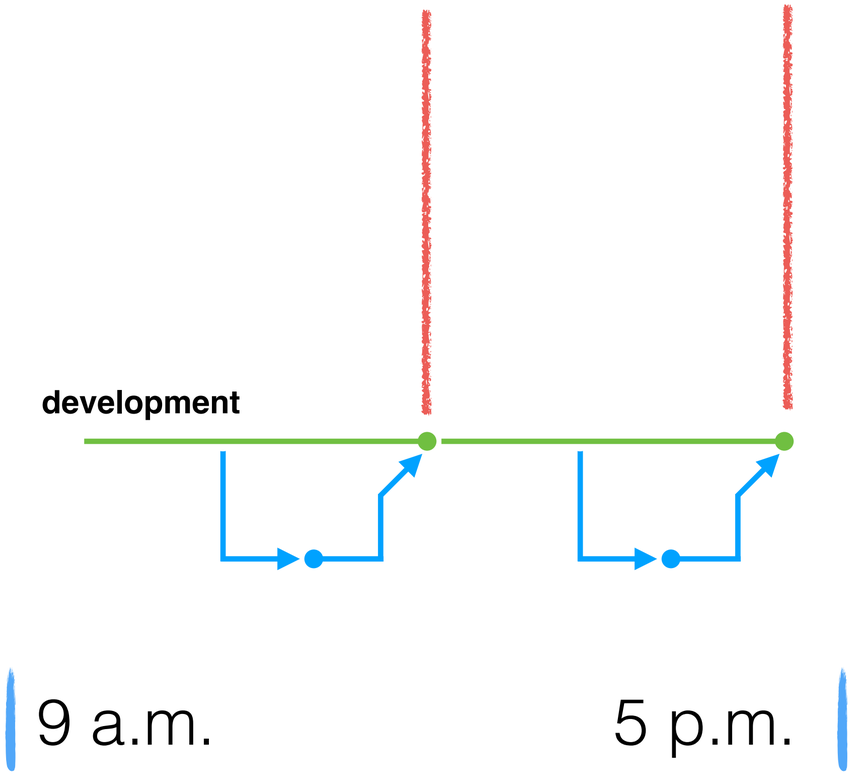
did we regress?

are tests failing?

being hasty does that

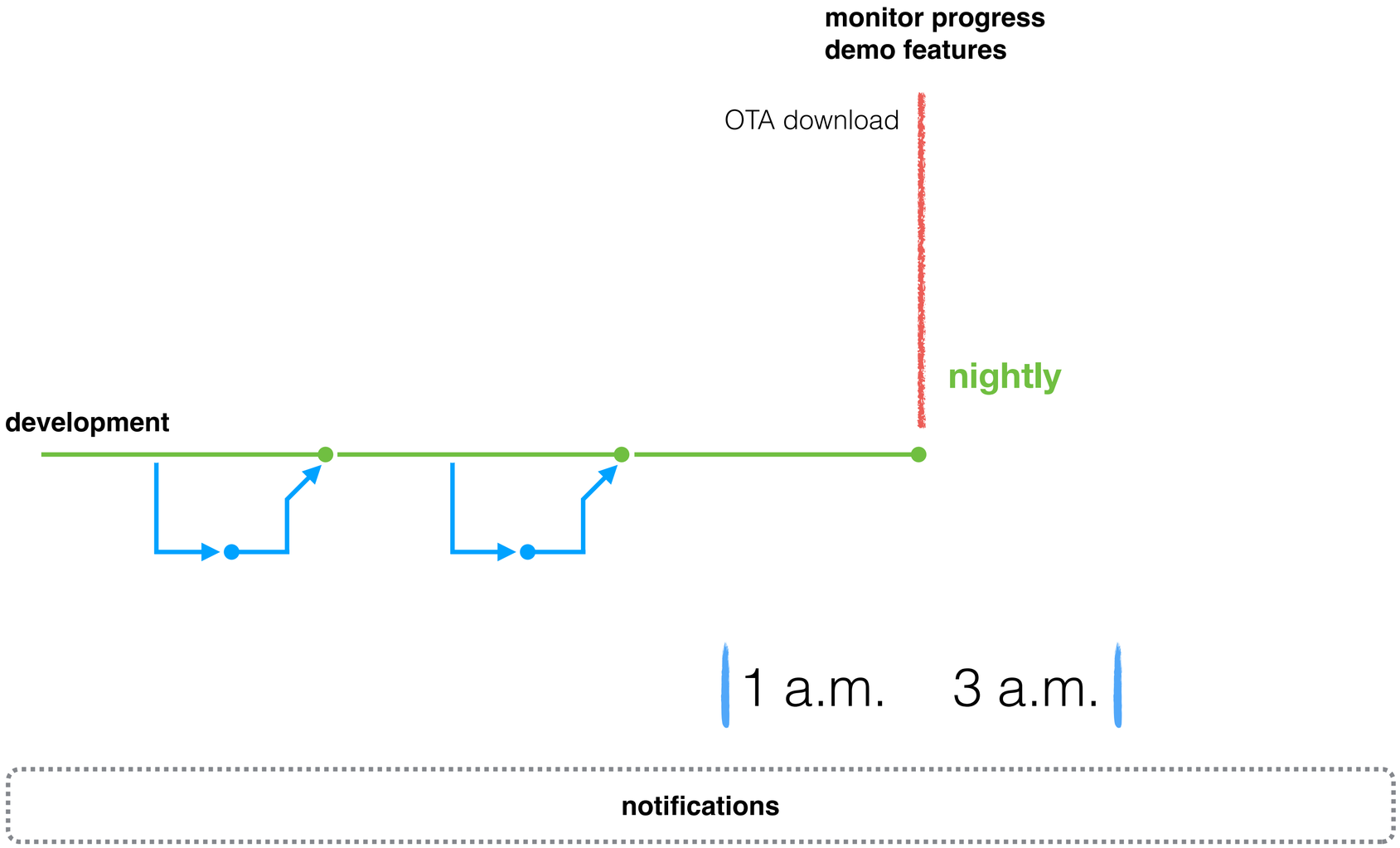
so a day goes like this

nine to five


development
9 a.m.
5 p.m.

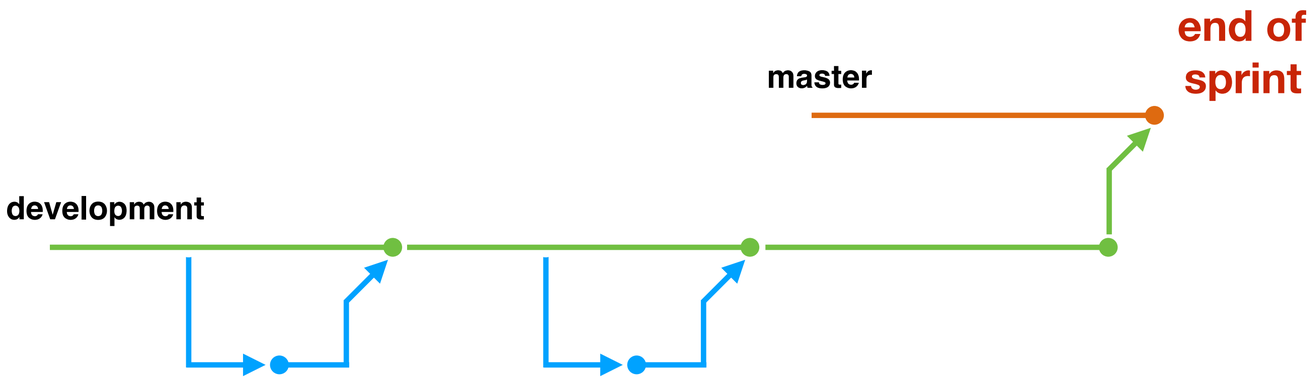
hitting a beat

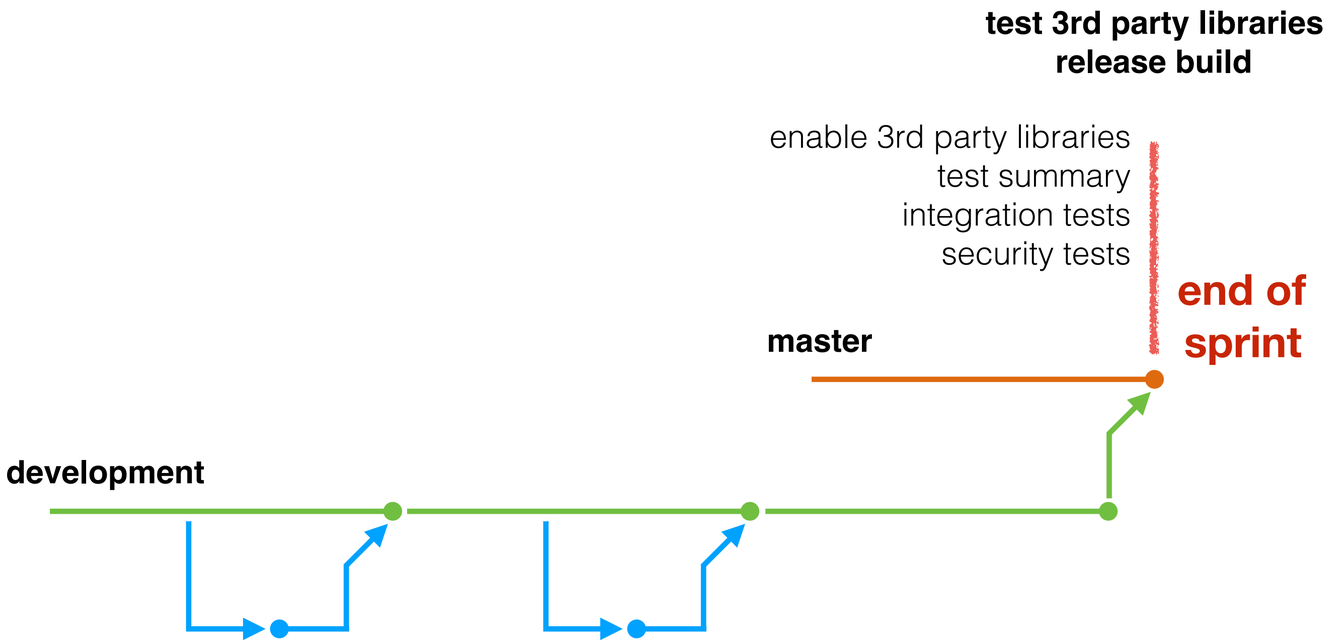
making progress


monitor progress
demo features
OTA download
nightly
development
1 a.m.
3 a.m.
notifications

demo day, every week

monitor progress


end of
sprint
master
development


test 3rd party libraries
release build
enable 3rd party libraries
test summary
integration tests
security tests
end of
sprint
master
development

enable 3rd party libraries

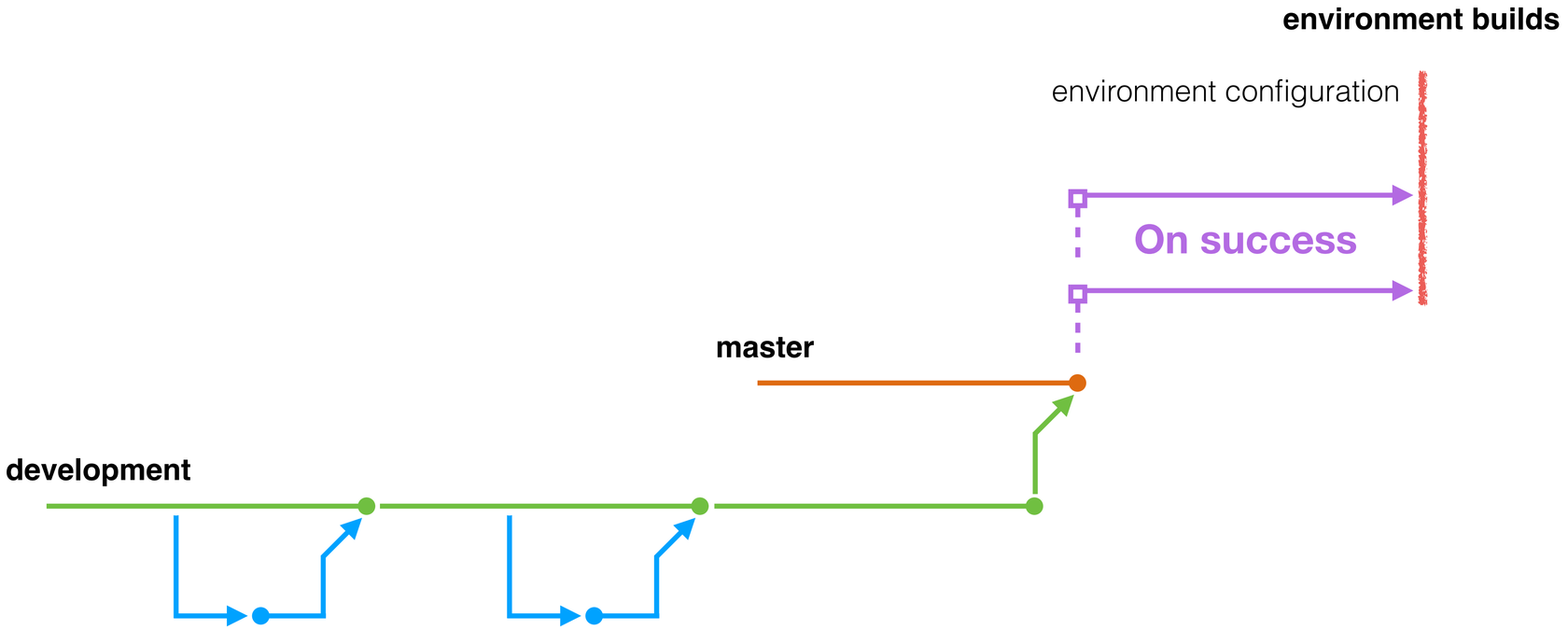
jailbreak, code obfuscation, anti tampering, etc.

integration tests

against a staging server

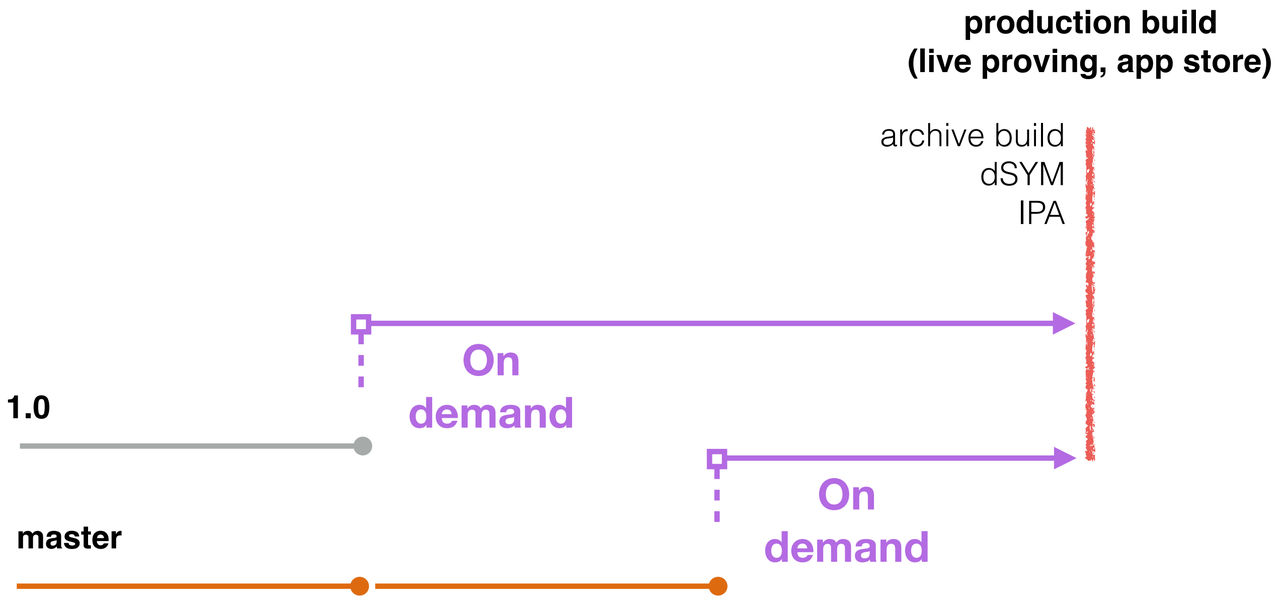
security tests

against a device


environment builds
environment configuration
On success
master
development

environment builds

environment configuration (e.g. SSL)


production build
(live proving, app store)
archive build
dSYM
IPA
On
demand
1.0
On
demand
master

live proving

and it’s off… for weeks

the technical details

a big pile of scripts and ideas

time

and effort

a script describing each build stage

a configuration if you like

Jenkins setup

too much involvement

development

xcodebuild -scheme *hello-world* -configuration Debug ***clean build test***

nightly

xcodebuild -scheme *hello-world* -configuration Release -destination “generic/platform=iOS"
archive -archivePath *hello-world.xcarchive*

nightly

xcodebuild -**exportArchive** -archivePath *hello-world.xcarchive -exportPath hello-world.ipa* -exportOptionsPlist exportOptions.plist

environment builds

resource substitution

Staging

environment

~jenkins/environments

a list of environments per project

git ls-tree master \
--name-only “hello-world”

a list of environments

```
git clone -b master \
~jenkins/environments/$1
```

$1 = project name repository, e.g. “hello-world”

/development
/staging
/production

replace files

just a copy

server.plist

resource substitution

SSL Certificate

resource substitution

build settings

conditional compilation

build settings

i.e. hello-world.xcconfig

compiler flags

OTHER_SWIFT_FLAGS = $(inherited) -D SSL_PINNING

-D SSL_PINNING

hello-world.xcconfig

#if SSL_PINNING

conditional compilation

-xcconfig

xcodebuild -scheme *hello-world* -configuration Debug
clean ***build*** *-**xcconfig** hello-world.xcconfig*

~jenkins/configurations

support multiple releases per project

```
git clone -b hello-world-1.0 \
~jenkins/configurations/hello-world
```

support the *hello-world-1.x* branch

jenkins agents

distributed building for free

https://wiki.jenkins.io/display/JENKINS/Distributed+builds

use tags

to distinguish xcode installations

a scheduler

select the correct configuration
given a project name and a branch

a distributed build system

to scale

What support did it provide?

with measurements or otherwise

3 teams

across 3 projects

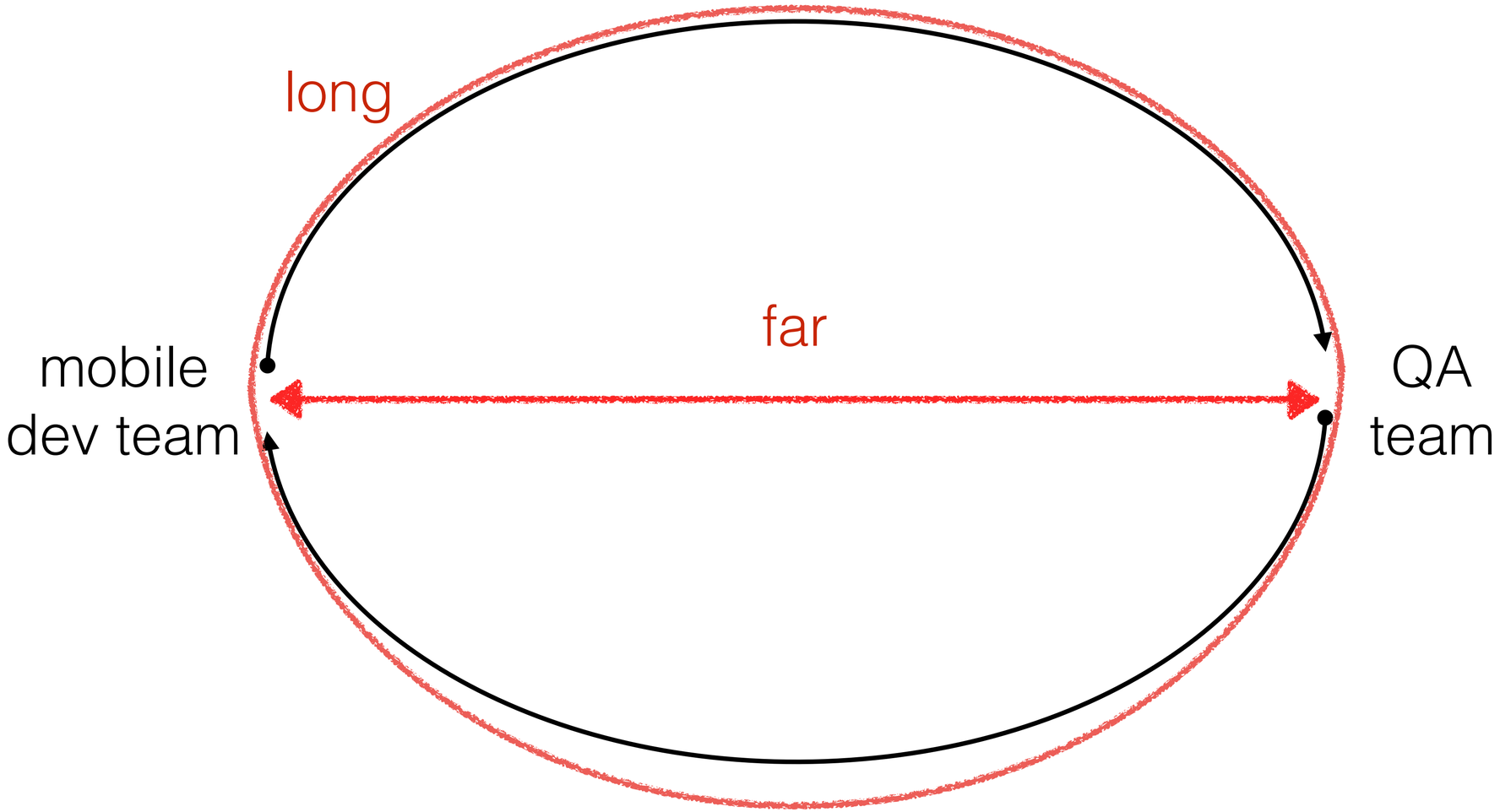
automated builds/releases

- 20 mins to deliver across all environments
- 15 mins to deliver to production
- Quality Gates (code coverage, tests run, security)

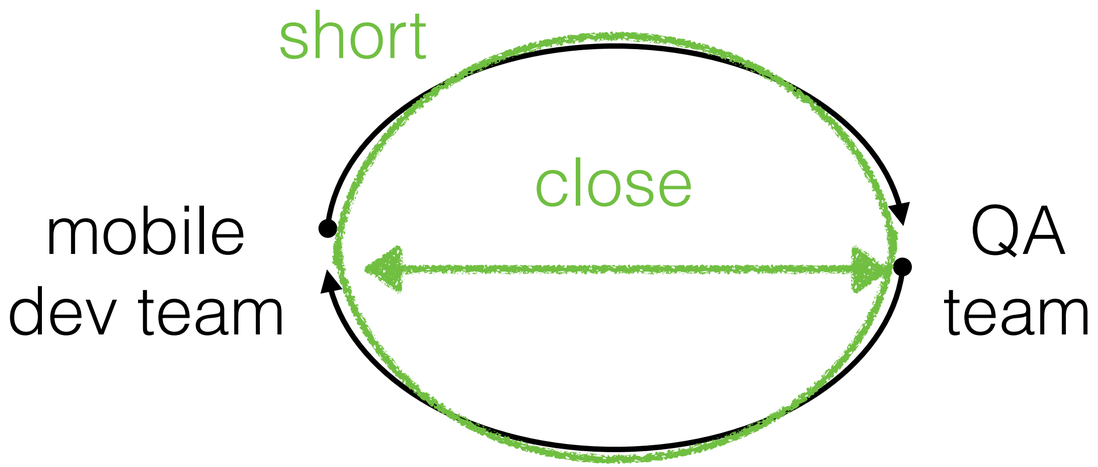
automated unit,integration tests

- 2168 unit tests in 18 seconds
- 33 integration tests in 2 mins 10 seconds

feedback loop



feedback loop



challenges

keep them in mind

consistency

across environments

“The Burden of Knowledge”

Craig Russell

https://medium.com/@trionkidnapper/the-burden-of-knowledge-52cc73508081

reproducing failures

locally

a set of scripts

obscure

disjointed user interface

jobs rather than pipeline*

* Jenkins Blue Ocean since then

not showing the full picture

what settings where used? what environments?

Future work

room for improvement

lots

- record user scenarios to play back for look & feel and catch regressions
- app should install and launch on every supported device/iOS version
- performance testing i.e. memory/CPU usage and trend.
- poor/no network connectivity scenarios. App shouldn't crash, should still be usable.
- tested on different cellular network operators, proxies, network configurations.

lots

- integration tests, spinning up “SIT” environments with a set of data
- accessibility. App should be accessible for people with disabilities.
- usability tests.
- battery drain.
- randomness. i.e user data, receiving a phone call while using the app, layout changes, localisation

Work of others

to help you go further

“Continuous integration for iOS with Nix and Buildkite”

Austin Louden | Pinterest engineer, Core Experience

https://medium.com/@Pinterest_Engineering/continuous-integration-for-ios-with-nix-and-buildkite-ef5b36c5292d

www qnoid.com