# qCon SF 2014 – Conference Report

Rolfe Bozier

18-Mar-2015

## Agenda

- What is qCon?
- Aside – why go to an SE conference?
- Interesting talks
  - “Continuous Delivery For the Rest of Us”
  - “Programming Should Be More Than Coding”
  - “How DevOps and the Cloud Changed Google Engineering”
  - “Designing the Second Interface”
- Some other common points

## What is qCon?

- 8 annual conferences around the world
- Practitioner-based rather than academic
  - No papers, just the talk (videoed) and slides
- 3 days of talks arranged in 6 topic streams
- Topics are near cutting-edge, but in actual use, e.g.
  - Big/streamed data processing, functional languages, continuous delivery, application deployment, culture, etc…
- Presenters are engineers from many successful companies
  - Twitter, Ebay, Google, Pinterest, LinkedIn, Tumbler, Etsy, Netflix, Facebook, Paypal, Amazon, …
  - And lots of smaller ones as well
- qCon SF 2014 – 3 days, 6 sessions/day x 6 streams, over 1100 attendees (sold out)

## What is qCon?

- Backed by InfoQ web site
  - News, articles, books, videos etc.
  - Anyone can sign up, subscribe to content feeds
- After each conference, videos and slides are made available to the public over a 6 month period
  - A lot of the videos from qCon SF 2014 are available here: [www.infoq.com/conferences/qconsf2014](www.infoq.com/conferences/qconsf2014)
  - If you want access to an unavailable video and can't wait, let me know and I can send it to you

# Why go to a Software Engineering Conference?

- Why should you *avoid* an SE Conference?
  - “Software Engineering”
  - 2 x 15+ hours in an economy seat, multiple days of listening to people talk
  - Presenting the report when you get back
- Why should you *go* to a SE conference?
  - You *are* out of date
  - You will learn not just different technologies, but *better* ones

## Why go to a Software Engineering Conference?

- What will you get out of it?
  - More ideas in a few days than in a year sitting at your desk
  - Inspiration!
  - Exposure to a bunch of people who have created some cool stuff

    … but who aren’t really much different to you
- You might find yourself coming back wanting to change things…

*And all should cry, Beware! Beware!*
*His flashing eyes, his floating hair!*
*Weave a circle round him thrice,*
*And close your eyes with holy dread*
*For he on honey-dew hath fed,*
*And drunk the milk of Paradise.*

## Why go to a Software Engineering Conference?

- What will you get out of it?
  - More ideas in a few days than in a year sitting at your desk
  - Inspiration!
  - Exposure to a bunch of people who have created some cool stuff

    … but who aren't really much different to you
- You might find yourself coming back wanting to change things…

*And all should cry, Beware! Beware!*
*His flashing eyes, his floating hair!*
*Weave a circle round him thrice,*
*And close your eyes with holy dread*
*For he on honey-dew hath fed,*
*And drunk the milk of Paradise.*

GC has budget for 1 person to go to a Software Engineering Conference in 2015

## *“Continuous Delivery for the Rest of Us”*

- About the Guardian’s experiences managing website updates
- “Reduce the bottlenecks that stop you delivering”
- “Accelerate your OODA loop – get inside your competitors’”
- How?
  - Try doubling your release frequency (pretend if you need to).
- What are the blockers?
  - Monitoring is hard – automate your status indicators
  - Avoid merge hell – commit often, maybe with feature switches
  - Reduce manual QA
  - Automate performance analysis – graphs aren’t reliable
  - Pain points – the people feeling them may not be the people who can fix them

## *“Continuous Delivery for the Rest of Us”*

- How is this relevant to CiSRA?
- Release cycle ≈ 1 year (or more)
  - What if it was 6 months (or 3 months)?
    - We’d get a lot better at it
    - Maybe Canon could deploy the software earlier
    - We’d have fewer surprises

## *"Programming Should Be More Than Coding"*

- Keynote from Leslie Lamport (distributed systems, $\LaTeX$, TLA+)
- We should spend more time thinking before coding, i.e. writing specifications.
  - But how?  Borrow from maths/science
- TLA+ (Temporal Logic of Actions) is a tool to help with this
  - Break down algorithm into a series of init-state / next-state definitions
  - Define behaviours using:
    - Safety properties (nothing bad happens)
    - Liveness properties (something good happens, eventually)
  - Run a model checker over the TLA+ specification to verify it
- Used by Amazon AWS team to check their designs

## *"How DevOps and the Cloud Changed Google Engineering"*

- Keynote from Melody Meckfessel (Google Engineering Director)
- Google Cloud computing
  - IaaS – Google provides network, storage, servers, virtualisation
  - PaaS – IaaS + O/S, middleware, runtime
  - SaaS – PaaS + data, applications
- Cloud platform uses the same platform as Google itself
- DevOps: 800K builds, 100M+ tests, 30K+ changes – each day
  - Single repository
  - Dogfooding

## *“How DevOps and the Cloud Changed Google Engineering”*

- Testing is seen as extremely important
  - Test the whole codebase after every commit
  - Make testing available before committing
  - Easy test environment management
- “Testing has our back”
  - Merge early and often
  - Easy experimentation and rollback
  - Try ideas, and maybe abandon them
- When things go wrong
  - Logs are aggregated – only one place to search
  - Performance tracing is always on
  - Analysis / debug / investigation tools are all integrated

## *"Designing the Second Interface"*

- The brain relies on pattern recognition to reduce load
  - Visual pattern recognition bypasses the normal path and is very fast
  - It is like adding extra cognitive bandwidth
  - UI/UX designers understand this
- As developers, we have some understanding of UIs and usability
- What about the interface between you and the code?
  - Syntax highlighting
  - Semantic highlighting
- Modern IDEs and editors usually have some syntax colouring
  - Should comments be made to stand out or de-emphasised
  - Should variables be one colour, or each one with its own?
  - Should == and = be different colours?

## *“Designing the Second Interface”*

- Emphasise the important and de-emphasise the irrelevant
- It might be nice to have different colour schemes for different tasks
  - Writing code
  - Reading / understanding code
  - Debugging code
- The CiSRA angle
  - Unfortunately Visual Studio / Visual Assist X have limited support for adjusting syntax colouring
  - But don’t go over the top
    - *“Everyone can be super! And when everyone’s super… no-one will be!”*

## Some other common points

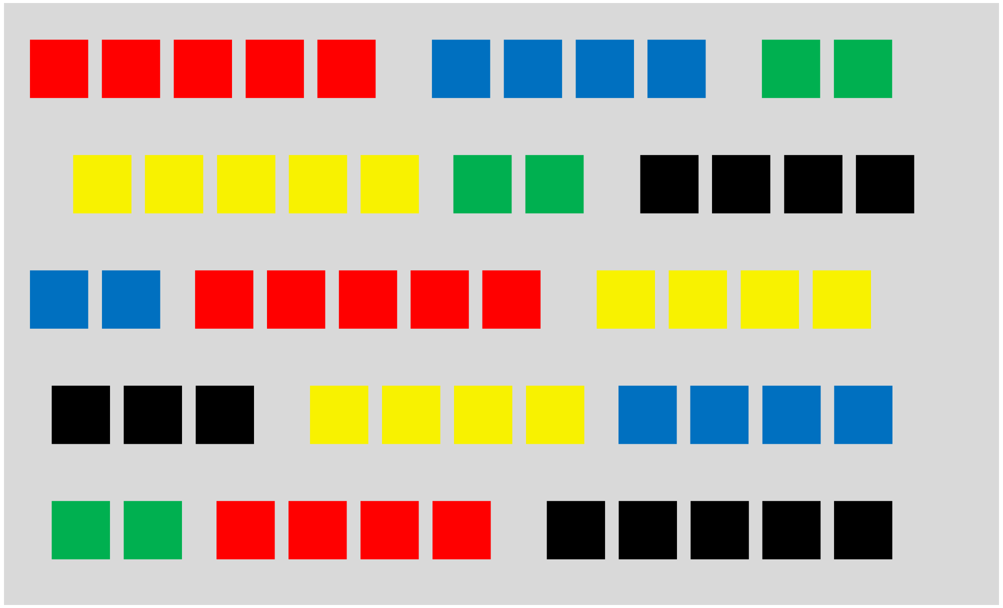
- Asserts – were mentioned by several presenters
  - Tend to be used for "error handling"
  - Should be treated as automatically-verified comments
    - Define pre-conditions, post-conditions, invariants
    - Should be everywhere
    - Enforce the specification in the code
  - In practice:
    - Amazon AWS – 99% of asserts are live in production code
    - Linux kernel contains 11,000 asserts – asserts are always on
    - Gcc and llvm compilers – asserts are always on
    - Valgrind – asserts are always on
    - Mars lander (NASA) – asserts are off ☺

## Some other common points

- Merging / committing to the trunk
  - *"Long-lived branches are good"* – no-one
  - Merge your code at least once a day
  - Long-lived branches are incompatible with continuous deployment
  - Complete testing only happens on the trunk
    - Your branch isn't getting this
    - No-one sees your changes until after you commit them

## Bonus – Visual pattern recognition

- The Stroop Effect
  - Name each of the colours



## Bonus – Visual pattern recognition

- The Stroop Effect
  - What is the colour of each word?

PURPLE YELLOW RED

BLACK RED GREEN

RED YELLOW ORANGE

BLUE PURPLE BLACK

RED GREEN ORANGE