Report from ICSE 2016

Hans Fangohr

2017-05-24

Outline

ICSE 2016

International Conference on Software Engineering (ICSE) 2016

- annual international 3-day meeting on Software Engineering *research*
- many satellite workshops for several days before and after the meeting
- registration fee relatively high
- 2016 meeting was in Austin, Texas (US) in May

Travel


Killeen
Bryan
Austin
Kerrville
San Antonio
Del Rio
Victoria
Piedras Negras
Beeville
ON/OFF
MENU

Venue

Catering

Catering 2

SE4Science

SE4Science


Case Study „Scientific Software"
Researcher at university develops simulation software in hot rolling mils domain, used by industry partner (voestalpine Stahl)
50k LOC of C++ source code, developed over 5 years by academic partner
Introduction of Rulebook Generator to stimulate documentation writing
Documentation expected (by industry partner) on three levels:
Level 3
Scientific models, key findings and references to scientific theory
Rulebook Generator
Level 2
Numerical solutions the implementation is based on
Level 1
Technical software documentation (e.g. API documentation)
Doxygen
s c c h
SE4Science

SE4Science

Attached Workshop: Software Engineering for Science (SE4Science)

`http://se4science.org/workshops/se4science16/schedule.htm`

Micromagnetic simulation tool – SE lessons learned:

- talk: `http://www.southampton.ac.uk/~fangohr/publications/talk/2016-05-16-ICSE-SE4Science-Austin-Texas-US.pdf`
- paper: `http://doi.acm.org/10.1145/2897676.2897677`

Continuous Deployment (CD)

One step back: Continuous Integration (CI)

Continuous integration

Repeated integration of new code into main line, including

- automatic builds
- automatic unit, integration and system tests

Common implementation

- git for version control
- push changes to github (merge request)
- triggers running of test suite (Travis CI)

Paper: Continuous Deployment at Facebook and OANDA

Tony Savor *et al*:

Continuous Deployment at Facebook and OANDA

DOI: `http://dx.doi.org/10.1145/2889160.2889223`

ICSE '16 Companion, May 14-22, 2016, Austin, TX, USA

What is Continuous Deployment (CD)

1. software updates are kept as small and isolated as reasonably feasible
2. they are released for deployment immediately after development and testing completes
3. the decision to deploy is largely left up to the developers (without the use of separate testing teams), and
4. deployment is fully automated.

Uptake of CD

Continuous deployment has been embraced by a number of high-profile Internet firms:

- Facebook was utilizing continuous deployment in 2005.
- Flickr reported 10 software deployments a day in 2009
- over 11,000 software deployments in 2011 at Etsy
- newly hired software developers at Etsy are assigned a simple bug to find and fix on their first day of work, and are expected to deploy their fix to production servers within a day or two – without supervision and without a separate testing team.

Values of CD

- small updates reduce risk
- engineers are given autonomy and end-to-end responsibility
- significant investment in tools
- encourage risk taking with a no blame culture

This paper

Quantitative and qualitative analyses of the continuous deployment practices at two very different firms:

- Facebook has thousands of engineers and a set of products that are used by well over a billion users; its backend servers can process billions of queries per second.
- OANDA, the second firm, has only roughly 100 engineers; it operates a currency trading system that processes many billion dollars worth of trades per day and is thus considered mission critical.

The continuous deployment processes at both firms are strikingly similar even though they were developed independently.

CD: Deployment Process

CD Principles

1. software is updated in small increments that are independently deployable
2. updates are the responsibility of the developers who created them
    - including being on call for failures

Broad responsibility results in short turn around time in the event of failures. Useful because

- developer still remembers what they have done
- there is a single point of contact

CD key practice 1: testing

1. Developer creates and runs automated
    - *unit* and
    - *subsystem* tests
2. Developer runs *integration testing* of the whole system on virtual machines
3. Automated *system tests* simulate production workloads
4. The *performance tests* are executed by the developers in non-virtual environments for reproducibility.

CD key practice 2: code review

- plays important role
- is accepted and taken seriously by developers because they are responsible for the full lifecycle of the software

CD key practice 3: deployment in stages

- Initially software updates are deployed onto a beta or a demo system.
- Beta/demo sites have real users and are considered production sites.
- Where possible, organizations use a practice commonly referred to as “dog fooding” whereby a portion of the development organization uses the most updated software before the changes are pushed to external users.
- Generally, the release of deployed software occurs in stages to contain any issues before general availability to the entire customer code base.

Staged deployment strategies 1/2

- dark launches: A deployment strategy where changes are released during off peak hours
- staging/baking: A stage in the deployment pipeline where a new version of software is tested in conditions similar to a production environment.
  An example of this is called shadow testing where production traffic is cloned and sent to a set of shadow machines that execute newer code than production. Results between production and shadow environments can be automatically compared and discrepancies reported as failures.

Staged deployment strategies 2/2

- blue-green deployments: A deployment strategy where a defective change to a production environment (blue) can be quickly switched to the latest stable production build (green).
  The change may initially be made available to, for example, 1% of the client base in a specific geographical location, thus limiting exposure (and with it, reputational risk), and only when confidence increases that the software is running properly is the fraction gradually increased, until it ultimately reaches 100%. When problems are detected, the fraction is quickly reduced to 0%.

CD: Results

Productivity

- Each Facebook developer releases an average of 3.5 software updates into production per week
- Each update involves an average of 92 lines of code (median of 33) that were added or modified.
- Use Lines Of Code (LOC) as proxy for productivity as the data is easily defined and was available
- Facebook team grew by factor 20 from 2008 to 2014
- Codebase grew by factor of 50 from 2008 to 2014

Productivity: LOC per developer constant

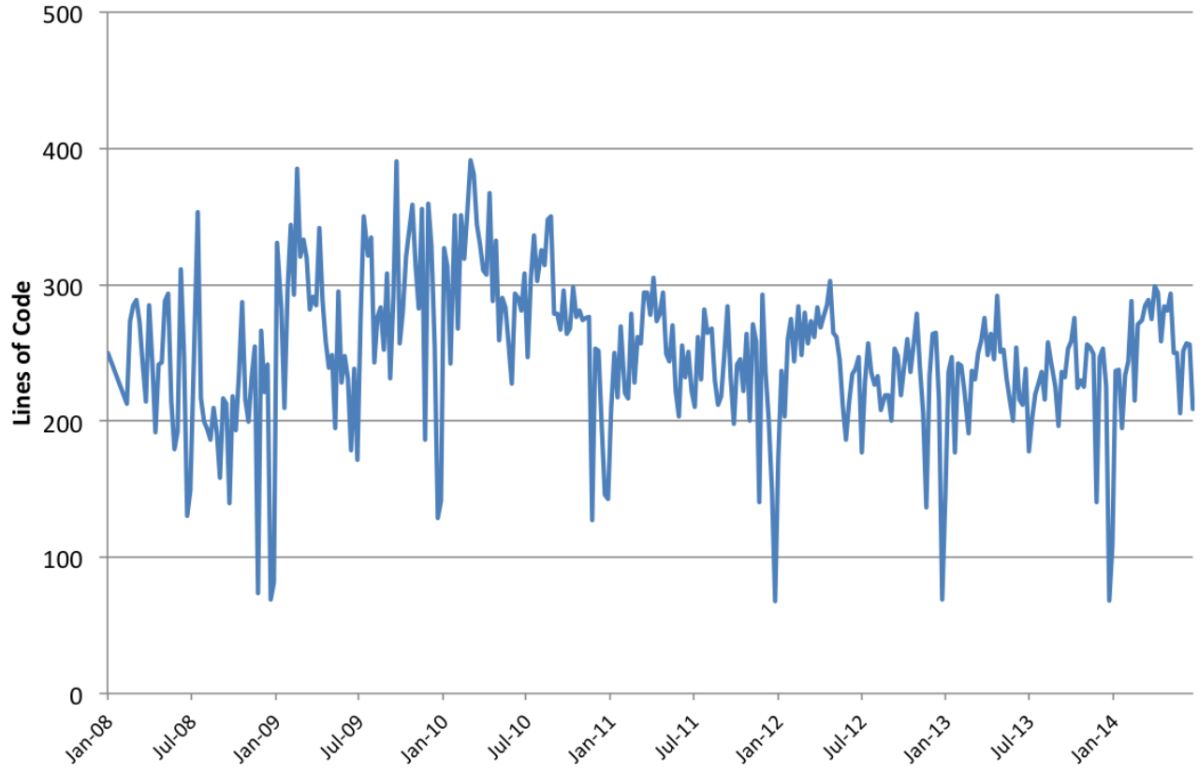


Figure 1: Lines of modified or added code deployed per developer per week at Facebook.

Quality

- Continuous Deployment process at Facebook has no testing team
- Qaulity relies on developers who are responsible for their code throughout the life cycle:
  - idea generation
  - architecture
  - design
  - implementation
  - testing
  - support in production
- Developers decide when to deploy their code and are also on-call

Does the quality of released code reduce as the team size grows?

Observation: number of critical issues was almost constant, regardless of number of deployments

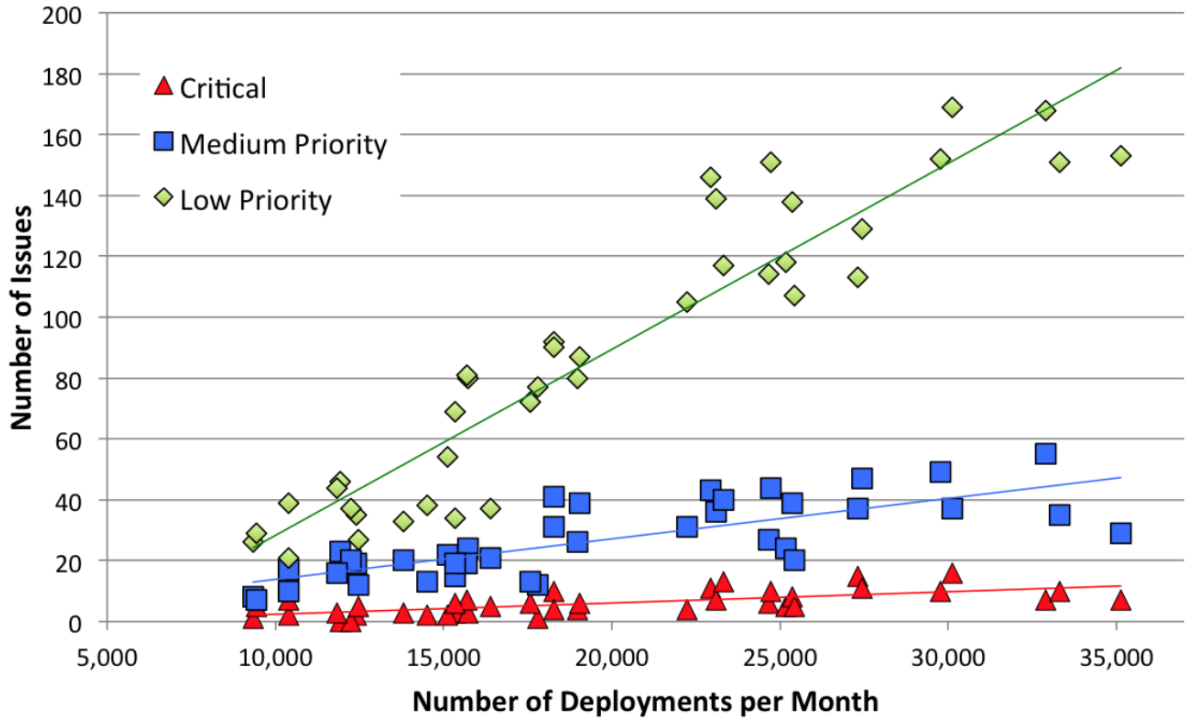


Figure 2: Number of production issues per month as a function of the number of deployments at Facebook.

Do developers prefer quick deployment into production?

- At Facebook by default code is released with next weekly release
- Developers can choose to put their code into daily release

Do developers prefer quick deployment into production?

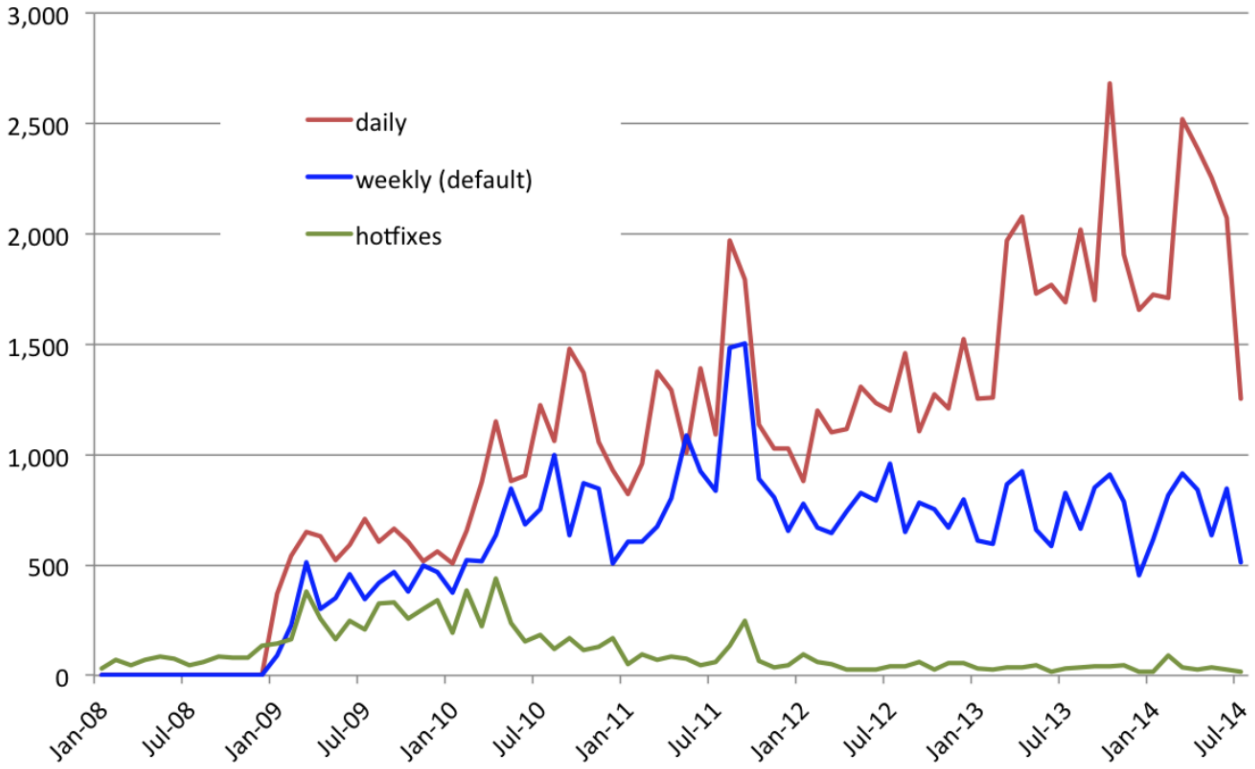


Figure 4: The number of deployment requests per week by type of request.

Human factors: leadership

Change in management:

- Management A: Nov 2010, engineering background and supporting continuous deployment.
- Management B: In mid 2012, management was replaced with executives having a business background. Their inclinations were more towards more traditional software engineering processes.
- Management C: At the end of 2013, executive management was replaced again. This team had a Silicon Valley background and was well versed with and supportive of continuous deployment.

One measure for efficiency is number of lines of code commit per week.

Human factors: management affects productivity

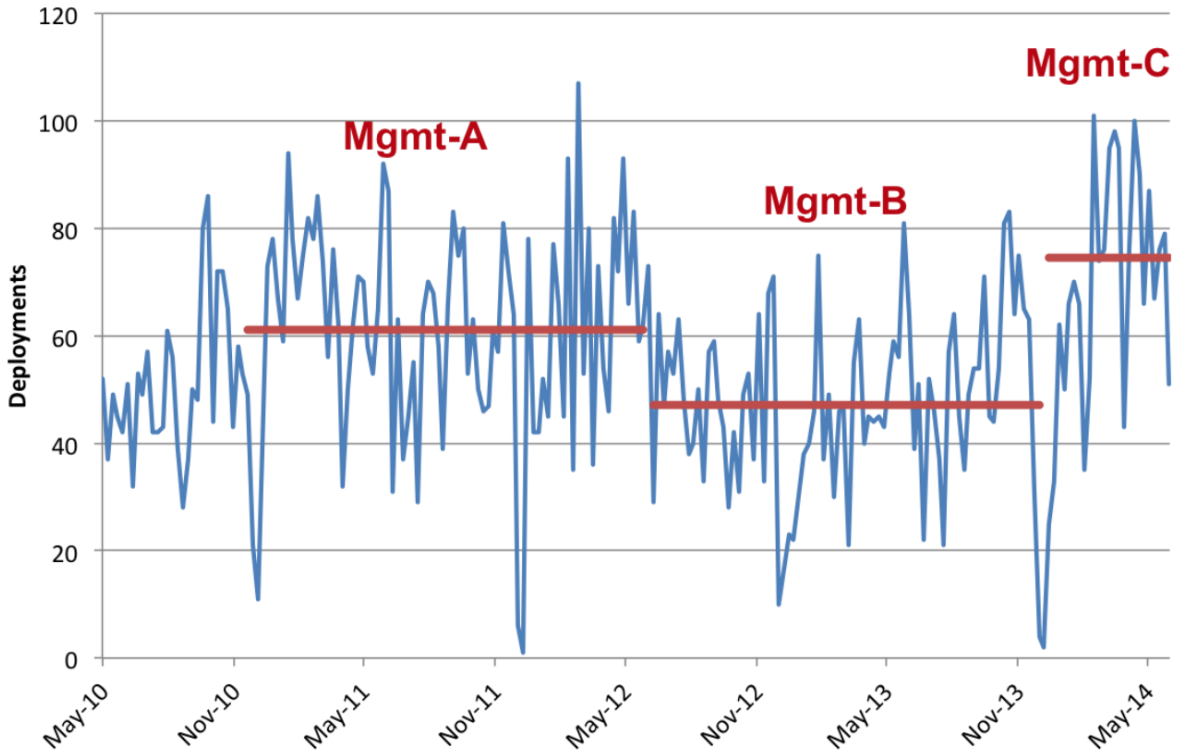


Figure 3: Number of deployments at OANDA per week over three management regimes. The red lines depict the average over the time periods shown.

CD: Other observations

Management approaches

Facebook: deal with large number of hotfixes

- weekly meeting
- no blame culture
    - not: what was the problem, but how could we do better?

Continous investment in tools

- automate and make repetitive work repeatable
- tools evolve
- off-the shelve often not good enough

Resource and performance creep is a problem

- aka *death by thousand cuts*
- small releases have an unnoticebale increase in resources (CPU, network, memory, storage)
- these add up
- difficult to monitor
- *performance testing* attempts to address this

Main results

1. continuous deployment does not inhibit productivity or quality even when the size of the engineering team increases by a factor of 20 and the code size grows by a factor of 50
2. management support of continuous deployment is vital
3. developers prefer faster deployment of the code they develop

Technical management essential

As with other agile methodologies, strong bottom-up culture with developers making many key decisions.

A different type of manager is needed because they play a different role: they influence rather than operate within a chain of command.

We find that in this environment, it is critical that managers be respected by the developers. Being technologically excellent makes this easier.

Filling management roles with suitable candidates has been a key challenge in both organizations. As CD is not widespread yet, management roles are often filled by promoting from within.

Chaos Engineering

Chaos Engineering Panel

Session on *<2016-05-18 Wed 16:05>*

`http://2016.icse.cs.txstate.edu/program/w4-5-seip`

Heather Nakama, Microsoft, United States Kyle Parrish, Fidelity Investments, United States Ian Van Hoven, Yahoo, United States Chris Adams, Uber, United States

and somebody from Netflix?

What is chaos engineering 1

- is another step in the testing timeline
- things go wrong in the real world
- how do you know how your system behaves when things go wrong?
- introduce those irregularities more regularly

Anecdote

CEO visits the data centre, and always pulled the first cable he could see. He wanted us to survive him.

Approach to deal with fear

If it is scary, do it so often until it is not scary anymore.

What is chaos engineering 2

Experimentation in the production system:

- fault injection
- network overload
- injecting latency (TCP latency)
- disconnecting two machines (TCP disconnects)
- poking at it, messing with it, see what happens
- see how the service behaves under really bad situations, under failures, while it falls apart
- see how (if) it comes up again
- do it in a safe environment
    - at Azure: not in systems used by customers

The Evolution of C Programming Practices: A Study of the Unix Operating System 1973–2015

The Evolution of C Programming Practices: A Study of the Unix Operating System 1973–2015

Diomidis Spinellis, Panos Louridas, Maria Kechagia

The Evolution of C Programming Practices: A Study of the Unix Operating System 1973–2015

DOI: `http://dx.doi.org/10.1145/2884781.2884799`

Unix source codes:
`https://github.com/dspinellis/unix-history-repo`

Overview of UNIX versions

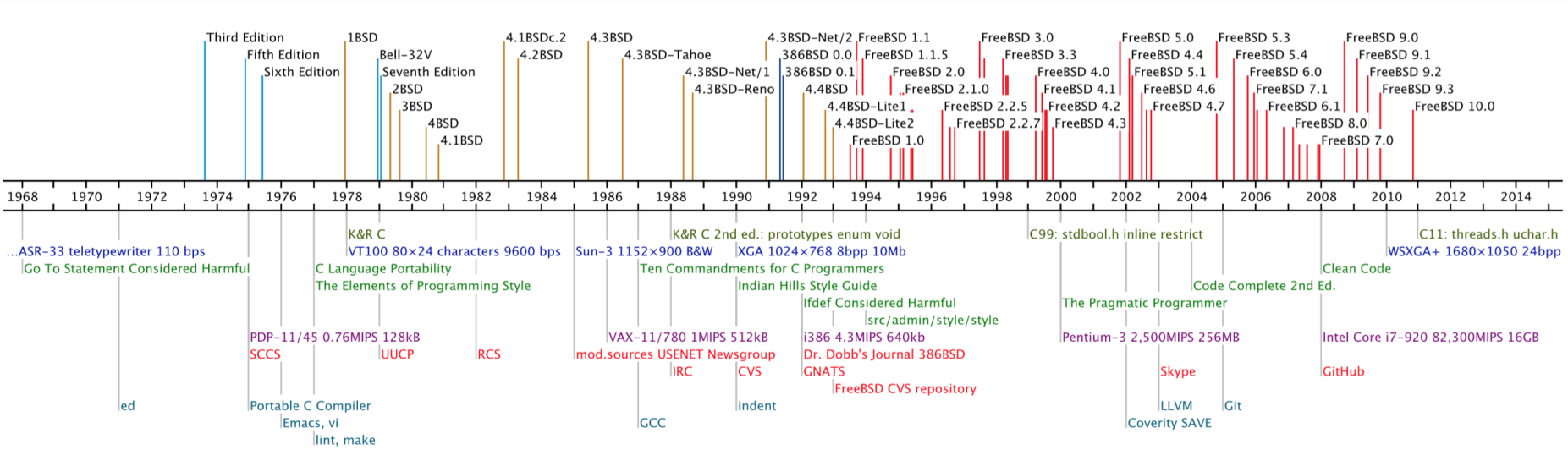


Figure 1: Timeline of indicative analyzed revisions and milestones in (from top to bottom): C language evolution, developer interfaces, programming guidelines, processing capacity, collaboration mechanisms, and tools.

Example `ed.c`

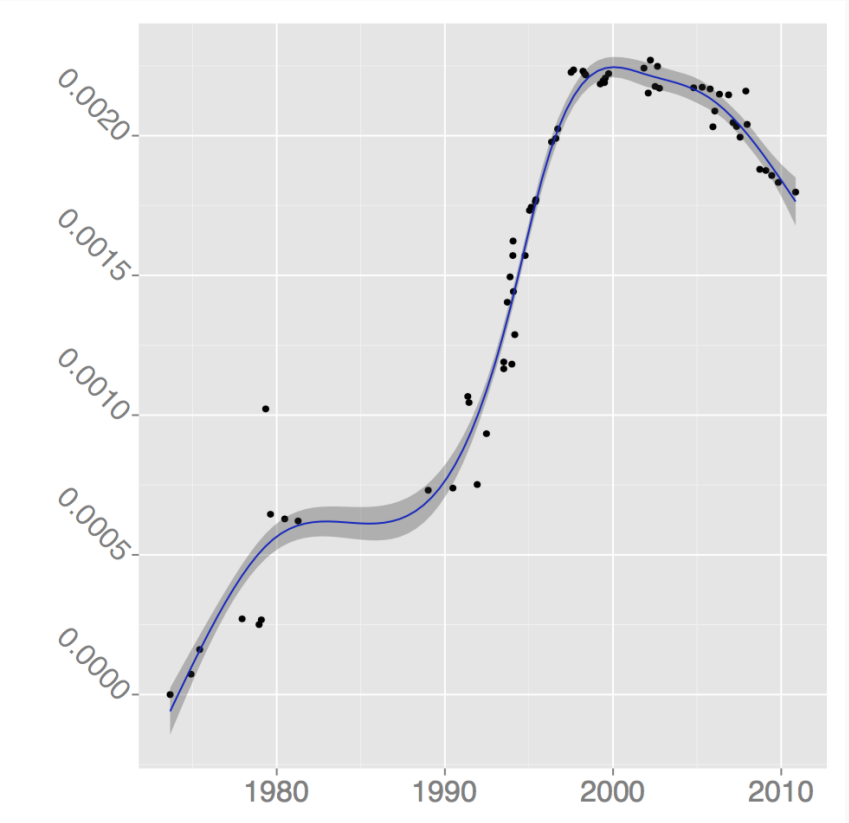
Copy of Unix source file from 1988:

68 global variabels, and one comment:

```
/*
 * Editor
 */
```

Kludge words

- sloppiness metric (*dkludge*): number of “kludge” words that may indicate problems in the code, including:
  - `fixme`
  - `xxx`
  - `todo`
  - `bugbug`, and
  - ”swearwords that cannot be reproduced in this paper”



Other comments

Assess the quality of a test suite through mutation analysis

Mutation analysis:

- change source code:
  - change operator
  - change order of operations
  - ...
- keep running test suite to see how many mutations are picked up

Test prioritisation

- developers like quick feedback
- throw all the resources you have at running the tests quickly (Microsoft)
- run tests that tend to fail first
- run new tests first

Summary

Summary

- interesting conference
- mixture of very fundamental research
- and reports and analysis of current and emerging practices

Full Programme available online:

`http://2016.icse.cs.txstate.edu`

- open community
- good catering
- joint lunch
- many volunteers helping