cphbusi

COPENHAGEN BUSINESS ACADEMY

Technical debt

.

Jens Egholm Pedersen
<jeep@cphbusiness.dk>

Recap

- Monitoring
  - Service-level agreement (SLA)
- Logging
  - Post-mortem analysis
- Scaling
  - Load balancing
  - Scaling
  - Monitoring of scaling
- Security
  - Threat modelling
  - Risk matrix
  - Pyramid of pain

cphbusiness

Goals of LSD

- Train the student to develop large-scale IT systems, where scalability is a key characteristic
- The student must have knowledge of concepts, techniques and technologies for the continuous integration and delivery of software-based systems
- The student must be able to design, implement, and maintain large distributed systems in distributed development teams

See also: Your curriculum 2017 (pdf)

Goals of the DevOps part

- Give you theoretical and practical knowledge on maintening and operating large systems

1) Monitoring	2. November
2) Logging	9. November
3) Scaling	16. November
4) Security	23. November

- Essentially everything that happens *around* the code

See also: Your curriculum 2017 (pdf)

Goals for today

- Guest lecture
- Assignment feedback
- Understand what technical debt is and how it can be avoided
- Gain practical knowledge on working with technical debt
- Helge: maintainability

Assignment feedback

- Generally nice work
  - You took a lot of time for this assignment

- Some of you don't put passwords on your DB
- Some of you store your passwords in plaintext
  - !!!!!111one

- You found some cool vulnerabilities

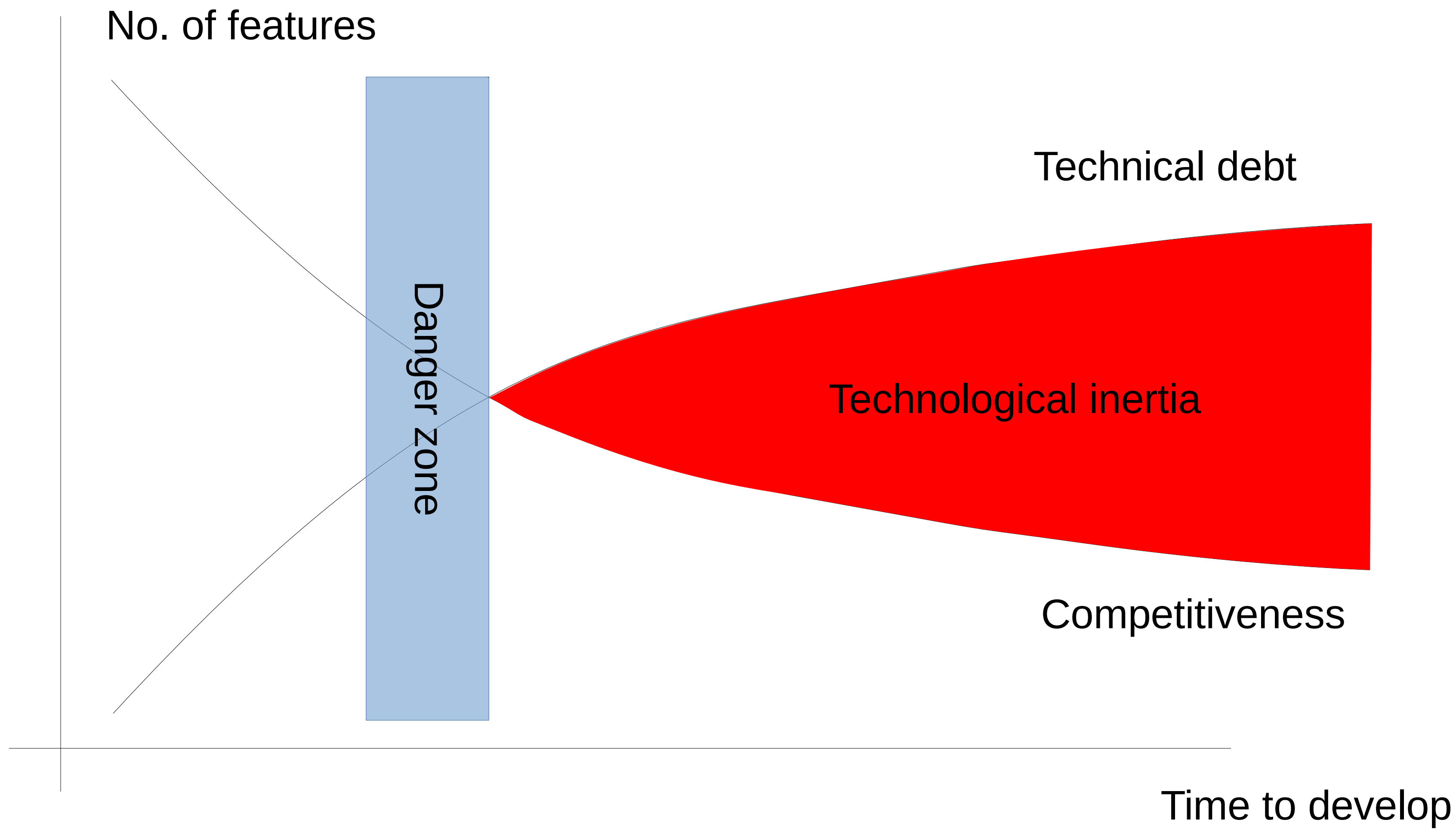
Technical debt

- Each feature adds complexity
- Complexity requires time
- Time requires money

- Humans are horrible logic machines

See also: Technical debt on Wikipedia

Technical debt



See also: Technical debt on Wikipedia

Avoiding debt

- Change avoidance
  - Avoid bringing yourself in a situation where you have to change
  - Get the requirements right
  - Design processes, RUP, agile etc.

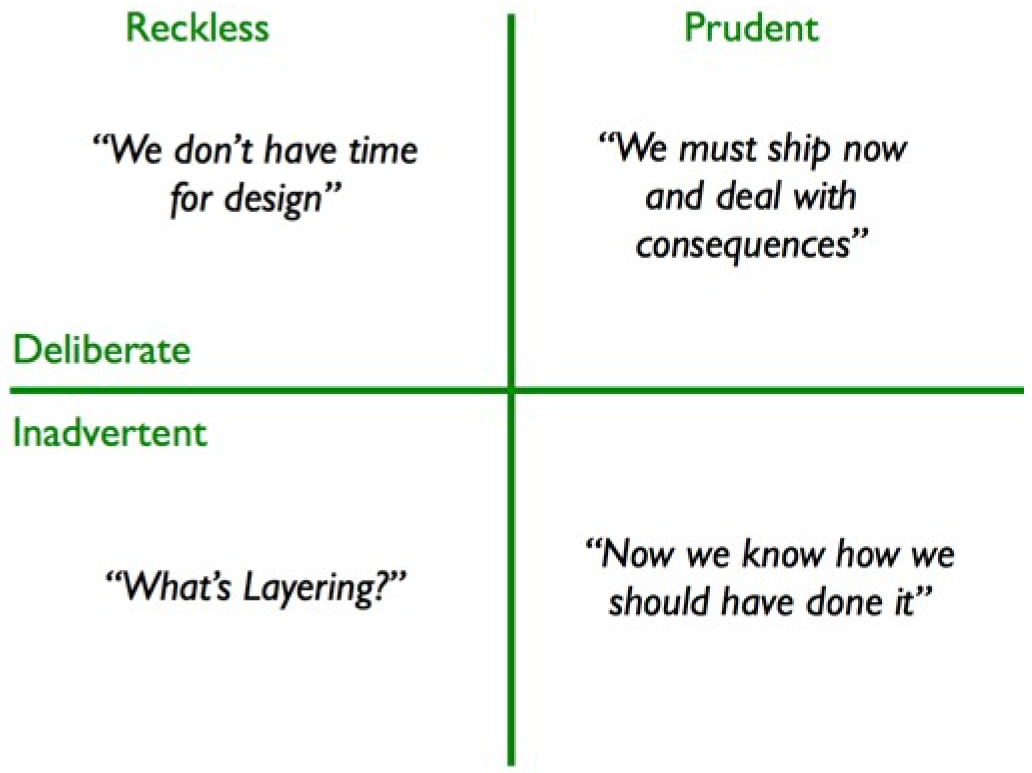
- Change tolerance
  - Design for change
  - Design patterns, compositionality, coupling etc.

How debt is introduced

- Debt can happen
  - Deliberately
    - “We don’t have time for design”
  - Inadvertently
    - “I’ll just re-implement this library function”
- And it can be caused by
  - Recklessness
    - “What is a design pattern?”
  - Prudent
    - “Fix now and deal with it later”

See also: Technical debt quadrant

How debt is introduced


Reckless
Prudent
“We don’t have time for design”
“We must ship now and deal with consequences”
Deliberate
Inadvertent
“What’s Layering?”
“Now we know how we should have done it”

See also: Technical debt quadrant

Working with debt

- Technical debt will be a fact of your life

- Read up on your code techniques

- New requirements
  1) Fix problem, increase debt
  2) Fix problem, reduce debt

- Long run
  1) Prepare for change
  2) Clean up regularly

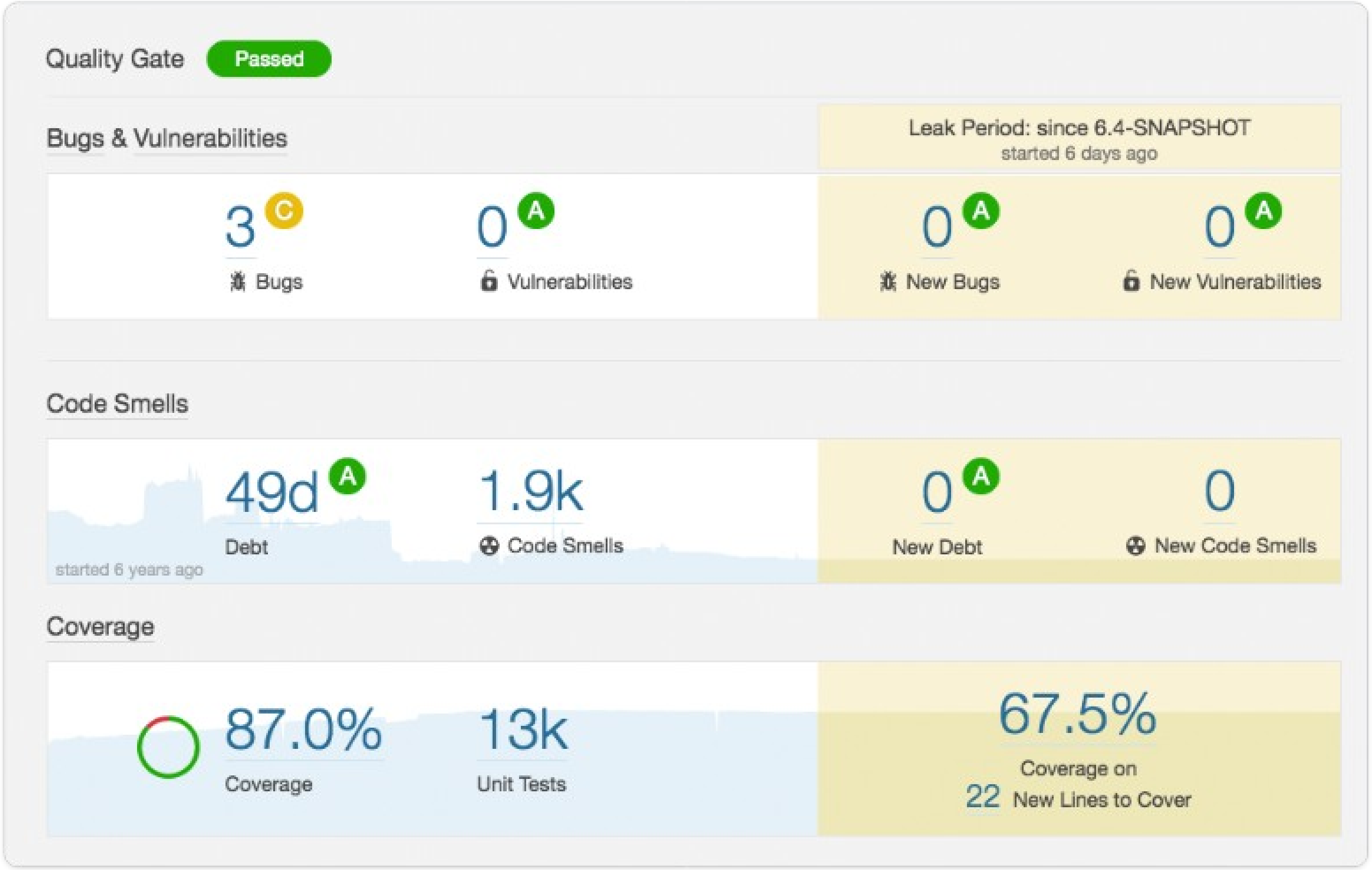
Measuring technical debt

- Metrics
  - LOC
  - Test coverage
  - “Code smells”

- CI
  - Measure debt difference
  - Refuse PR if debt increases

SonarCube

- Continuous code quality measurement: link



Recap

- Technical debt
  - Cost of change versus competitiveness over time
- Change avoidance
- Change tolerance