# TDD - RUNNING

## Strengthening the foundation

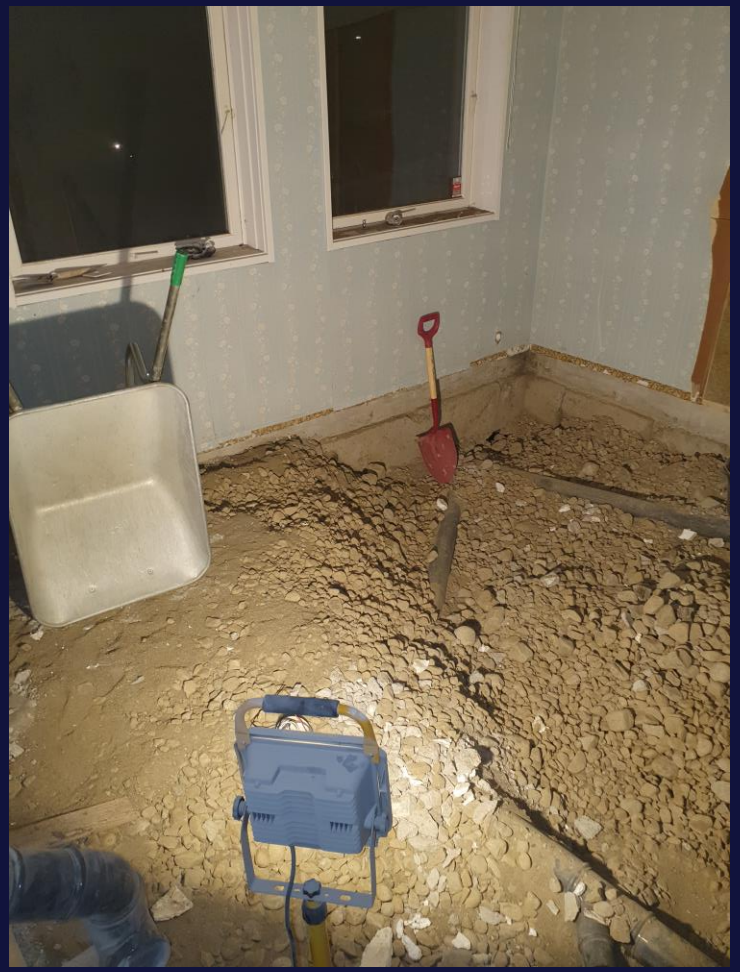
Kristoffer Steen

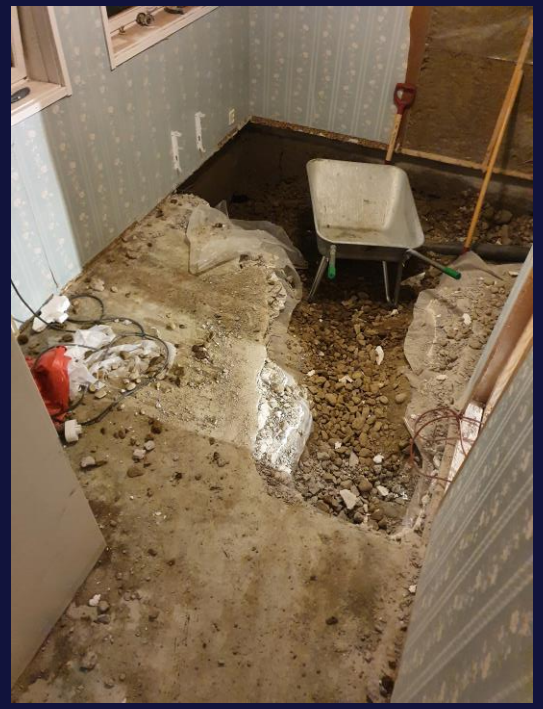
# Intro

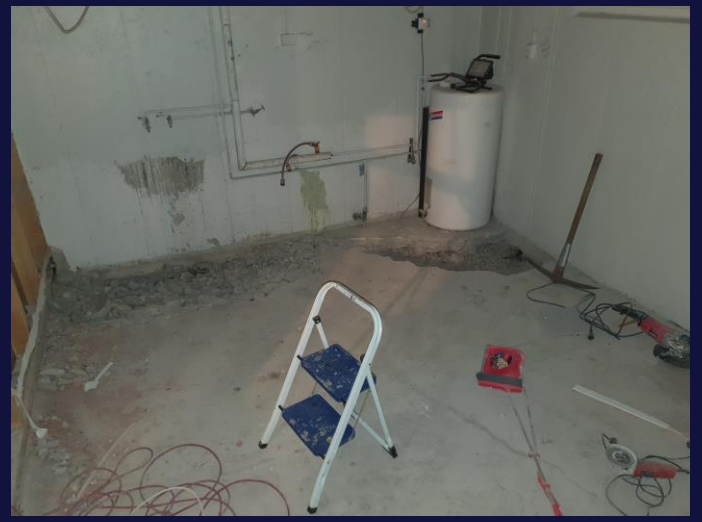
- The importance of a strong foundation
- SOLID principles revisited
- Refactoring
- Coupling and cohesion

bouvet

# The importance of a strong foundation

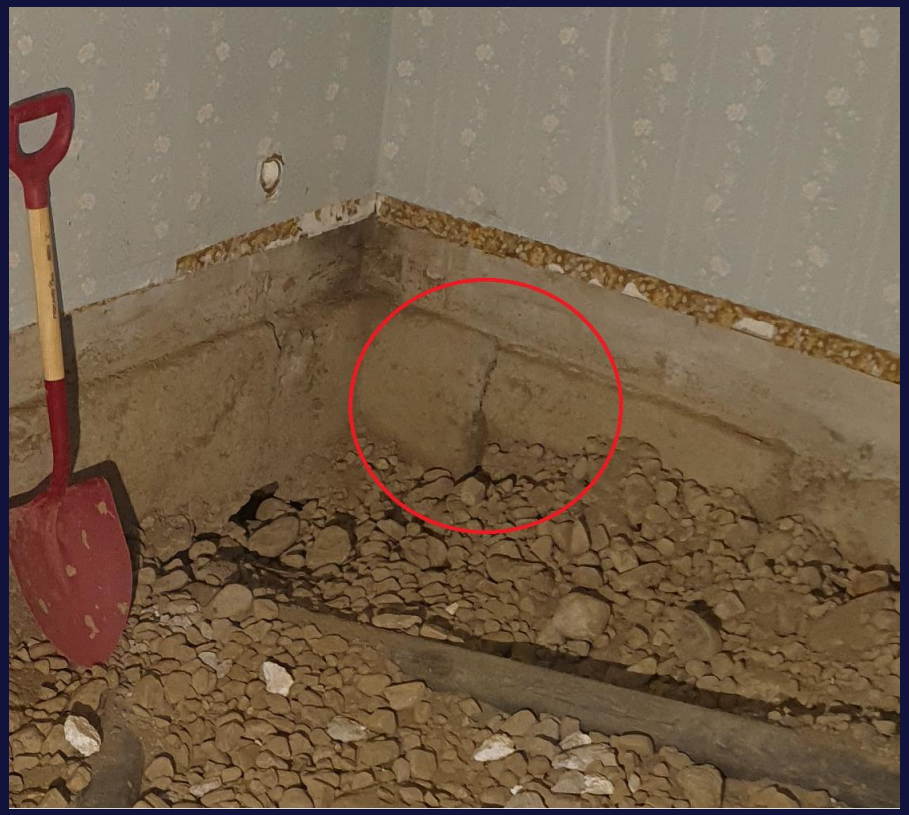






bouvet

# I found some cracks in my foundation



- Lack of vocabulary.
- Lack of code smell knowledge.
- Lack of pattern knowledge
  - ..or for some: The skill needed to apply them
- Lack of SOLID principle knowledge
  - ..or for some: The skill needed to apply them

GUILTY

bouvet

# Why are there cracks?

- It’s much more tempting and fun to learn new technologies and languages instead of reinforcing the foundation
- The stakeholders/customers want a house, they often don’t care much about the foundation. Especially when pressured.
- Because we are afraid to challenge other people, the tech lead or «the man» or whoever.
- Because we are afraid to take the road less travelled.




bouvet

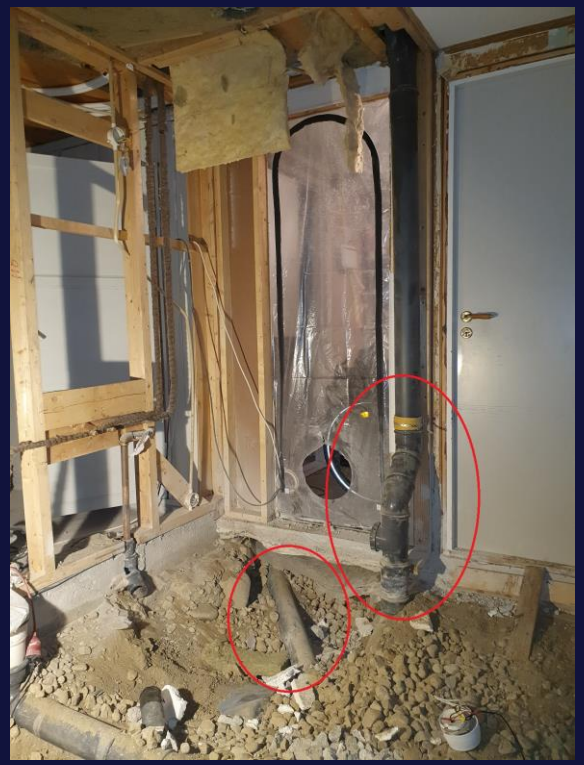
# SOLID revisited

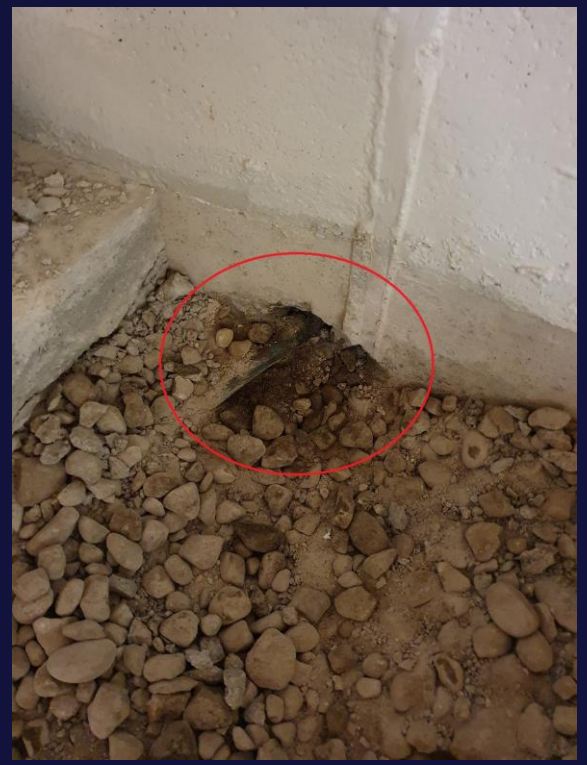
- Single Responsibility Principle
  - Modules / methods should have only reason to change
  - Smells: Large class, Long method, long parameter list, switch statements
- Open / Closed Principle
  - About modularity. Add functionality without changing existing code.
  - Goes against YAGNI
- Liskov Substitution Principle
  - Parent classes should be replaceable with subclasses without breaking code
  - Favour «has a»-relationships over «is-a»

# SOLID revisited cont.

- Interface Segregation Principle
  - Keep interfaces small, so many interfaces instead of few big ones.
  - Sort of like SRP for interfaces
- Dependency Inversion Principle
  - Both high level and low level modules should depend on abstractions
  - Abstractions should not depend upon details, details should depend on abstractions (contracts)

# Refactoring





- Stay in the green
- Don’t fix bugs in the exposed behaviour, clients might be relying on it

# Refactoring quick guide

- First, increase readability (counts for 80% of the improvement)
  - Do this layer by layer
- Then, refactor the design (counts for 20% of the improvement)
- To help stay in the green, use Parallel Change
  - Expand: «Add new code instead of changing existing code».
  - Migrate: «Allow clients to migrate to new code / client code point to new code»
  - Contract: «Remove deprecated code/tests»

# Coupling and Cohesion

- Coupling
  - How much interdependence between modules
  - Should be kept as low as possible
  - If high, making changes is hard

- Cohesion
  - How related the responsibilities within a class is
  - Should be kept as high as possible
  - If low, the class should probably be splitted

https://www.nzgeo.com/stories/squeezed-not-shaken/

# What this course tought me and how to take this further

- Improved skills and vocabulary
  - Easier to avoid/spot code smells, and easier to communicate them to others
  - Easier to spot and implement patterns and principles, and easier to communicate them to others
- I need to focus more on strengthening my foundation (fixing cracks)
- I should be honest about my cracks, we all have them
- I will keep doing kata’s, and i will be asking other convertees to join me
  - We have to organize some mob programming in our team/unit/company!

«The only easy day was yesterday»
US Navy Seals

bouvet

# Questions?

bouvet

# Thanks!

kristoffer.steen@bouvet.no

bouvet