EECE 310 – Software Engineering

Summary

Agenda

- Friday: Guest Speaker
- Evaluations
- Final Student Presentations (15 & 28)
- Course Summary and Q&A

Friday: Guest Speaker

- Title: **Upgradeability of 3d Party Libraries: A Security Perspective**
- Speaker: Prof. Arie van Deursen
- Part of final exam. Make sure you attend and ask questions!

Admin

- Course Eval: https://eval.olt.ubc.ca/apsc
- TA Eval: in-class

Instructor/Course Eval

- Go to https://eval.olt.ubc.ca/apsc
- Sign in and provide feedback
- It means a lot to me and the department!

TA evaluation

- Saba Alimadadi (F)

- Keheliya Gallaba

- Dawood Al-Masslawi

Notes for final

- Similar to mid-term (multiple-choice, true/false, open questions)
- 40% of the total course grade
- What to study:
  - All material covered in class
  - Lecture notes, **and your own notes**!
  - Reading Material on Connect
- Bring UBC card to exam.

Course Summary

The purpose

- Software **engineering** is not just programming (although that's a huge part)

- It is the process, the techniques/tools, the teamwork and the product.

Example: Facebook

- First written in PHP using MySQL by an undergrad (now a very wealthy undergrad!).
- Scale challenges: data storage and serving pages to billions.
- Now write PHP that gets compiles to C/C++ (HipHop)
- Haystack high-performance photo storage/retrieval system
- Data mirrored on geographically distributed content distribution networks (CDN).
- **SE huge part of this**: feature gathering, testing, release engineering, engineering tools, and writing code.

http://arstechnica.com/business/news/2012/04/exclusive-a-behind-the-scenes-look-at-facebook-release-engineering.ars

What we looked at

- Process (Waterfall, especially Agile: Scrum, XP)
- Versioning (especially Git)
- Requirements (especially user stories)
- Architecture and Design
- Cost Estimation

What we looked at

- Testing
- Code quality: smells and refactoring
- Software Evolution and Maintenance
- UI Design
- Crosscutting concerns and AOP

Prospects for an engineering discipline of software

- Software ‘engineering’ is a young field
- Challenges:
  - Rapidly changing environments
  - Science and engineering must reinforce one another
- Goals
  - Awareness of best practices
  - Measure/Find/Adopt disciplines that work!
  - Specialize and standardize

Software Process

- structured set of activities to develop a software system
- define roles in the process
- facilitates organization and communication

Process models

- **Waterfall**
  - define requirements up-front
  - hierarchical and traditional management approach
  - suitable for large projects (?) and new teams
  - low trust
  - Add iterations with Spiral model
- **Agile**
  - short iterations, focus on communication and change

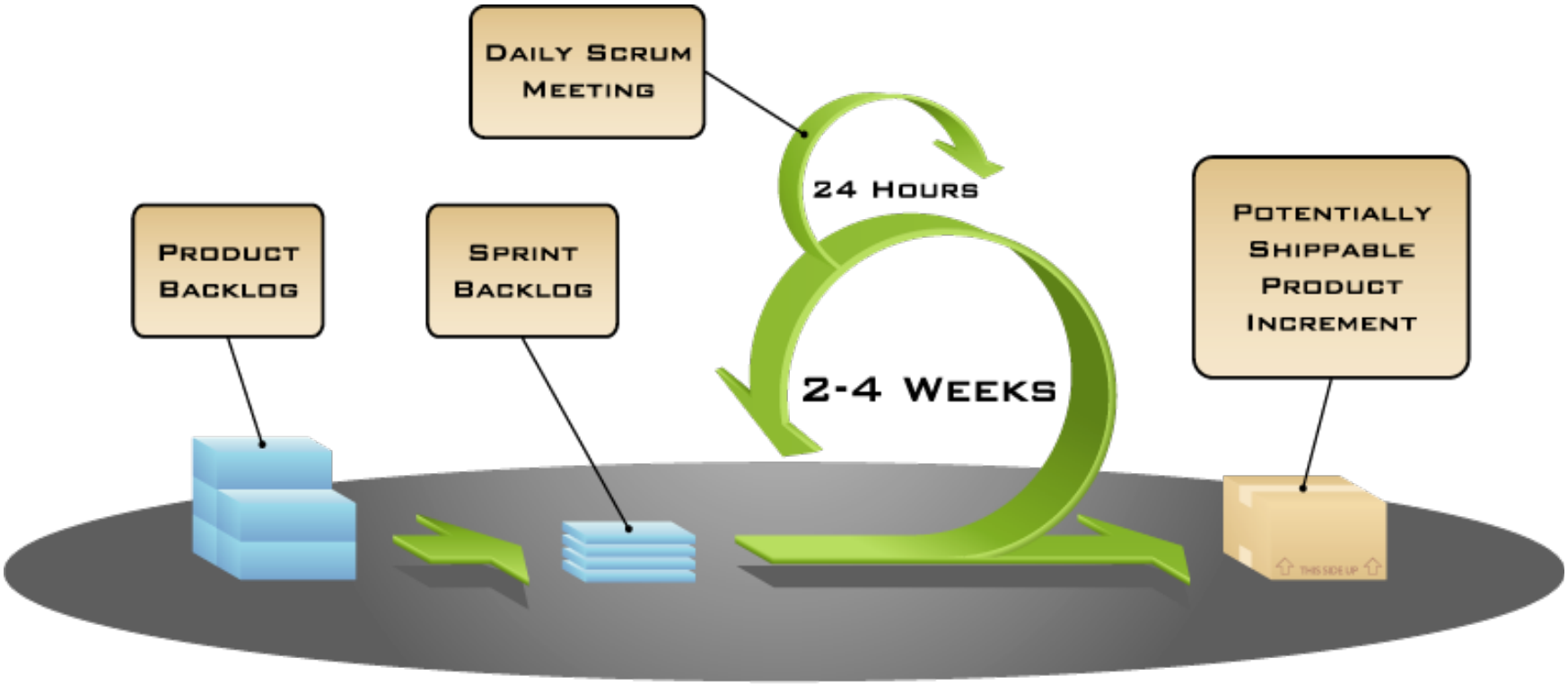
Waterfall: battleship, protected against everything that might happen…

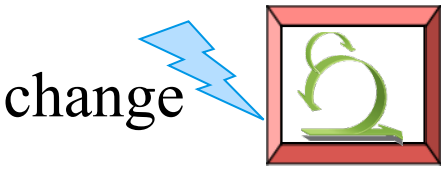
Agile: speedboat,
can change direction very quickly

Agile software development

- People over process
- Software over documentation
- Communication over contract
- Adaptable over plan-driven
- But, requires trust and capable team
- But, might require constant refactoring
- Agile methods include Scrum and XP

Scrum





Mountain Goat Software, LLC

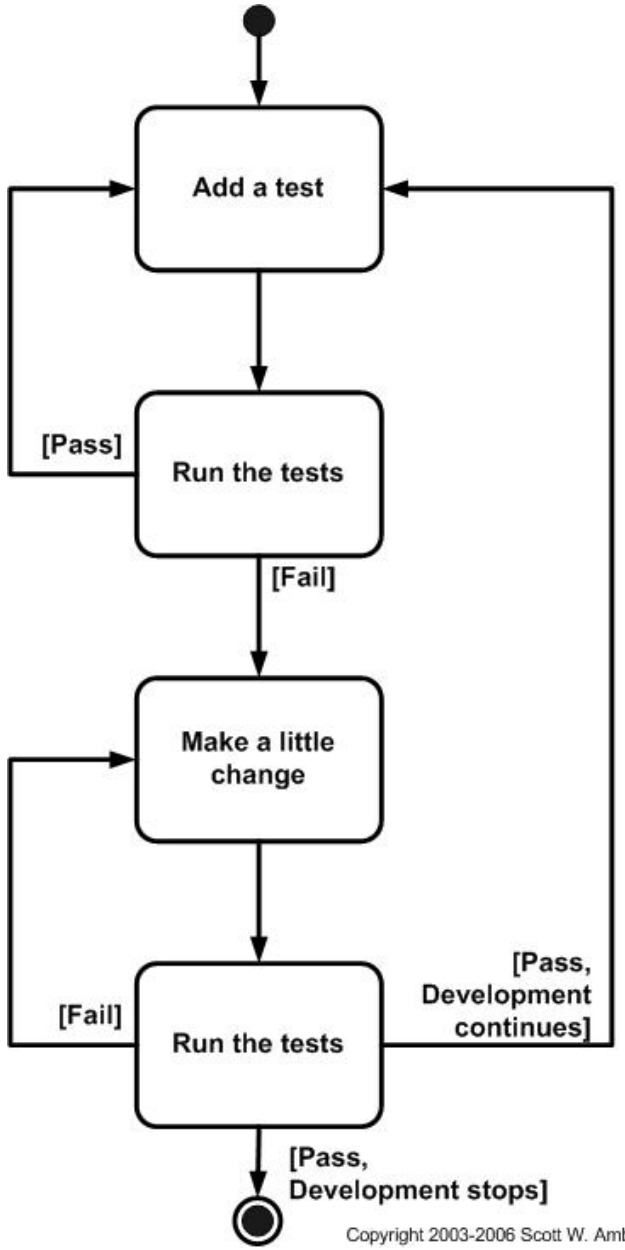
taken from www.mountaingoatsoftware.com (Mike Cohn)

eXtreme Programming

- User stories
- Unit Testing (xUnit)
- Test-driven development
- Continuous integration (eg. Jenkins)
- Pair programming
- Refactoring

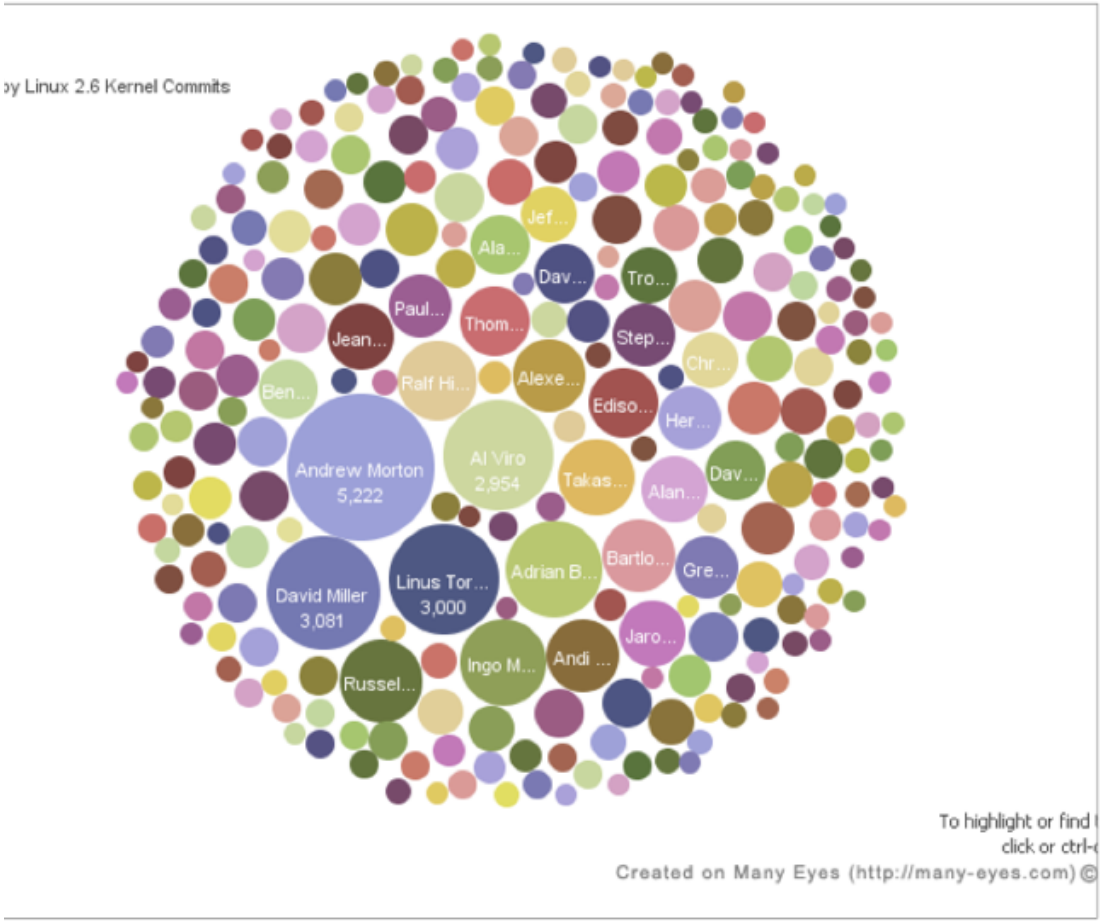
Test-driven development

- Tests define units of work
- Tests document expected functionality
- Tests are contracts for methods
- Later additions can be “regression” tested
  - Failure indicates breakage
- Small steps



Version control

- Why is it useful?
- Challenges
  - Central vs. distributed VC
- Merge and conflict
- Workflows
- Branching styles



Requirements: user stories

- As a [type of user], I want/can/need/etc. [goal or need], so I can [reason]
- has a description of the story providing additional detail
- specifies **acceptance criteria** that defines what is meant for this feature to be DONE
- provides **estimate** of the required effort (story points)

Requirements: motivation

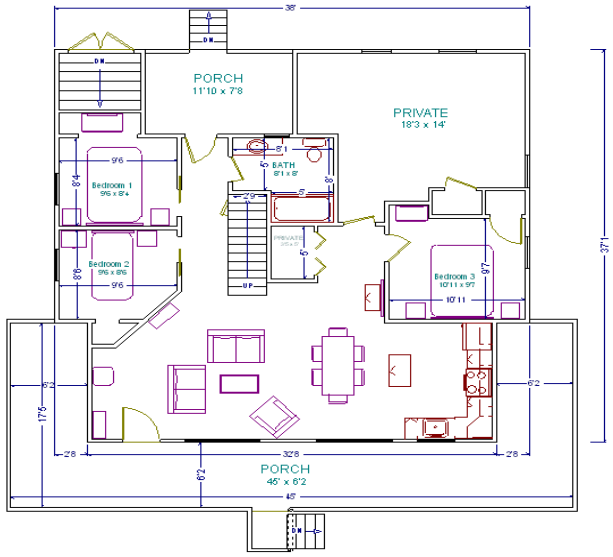
- Define *What* to build
- Define *Why* to build it
- Avoid describing *How* to build it
- Large source of costly errors
- Functional requirements linked to specific features of the system
- Non-functional requirements about wider system-wide properties (attributes, properties, constraints)

Requirements process

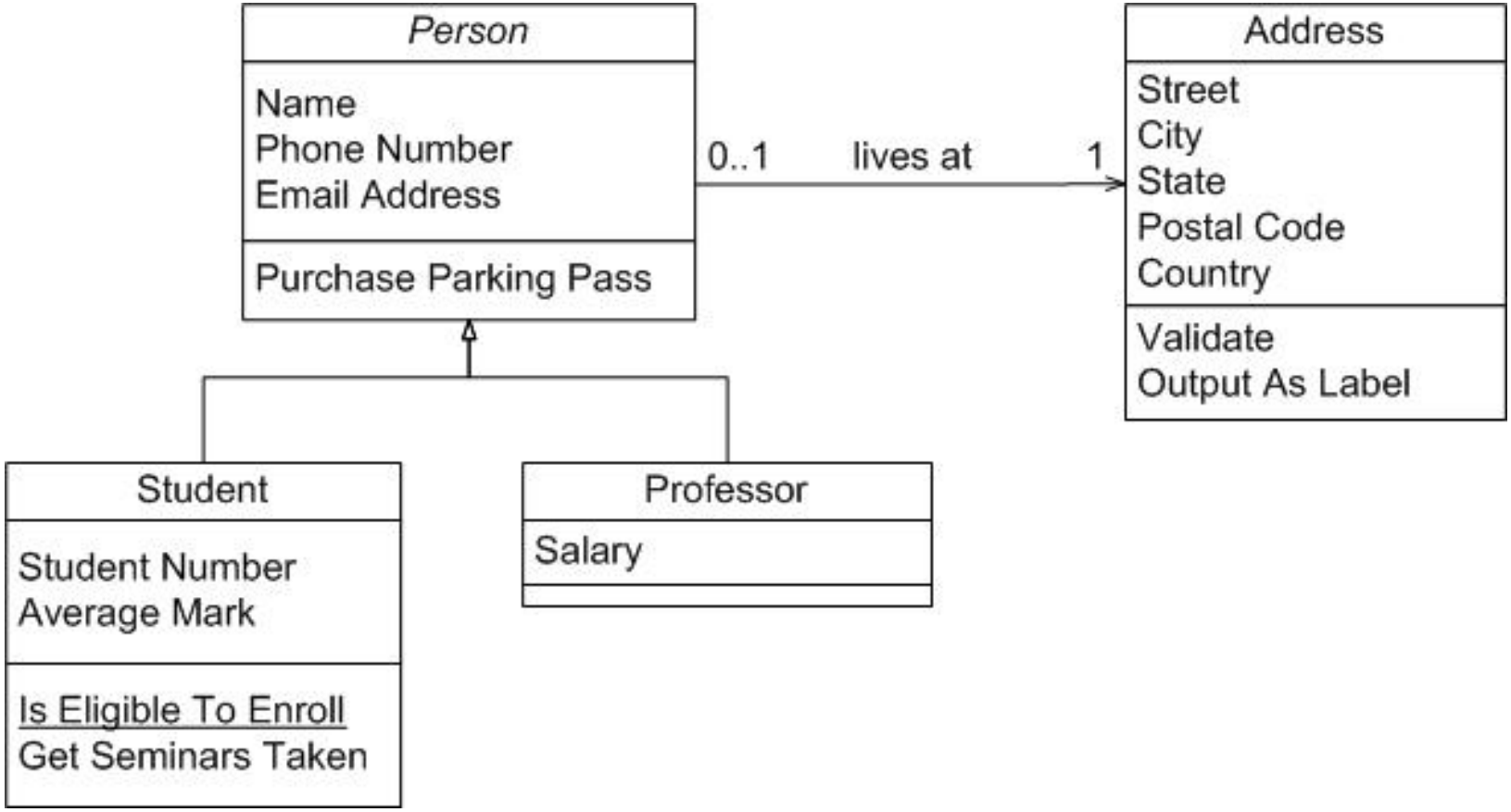
1. **Elicit** requirements by determining stakeholders, existing documents
   - Extract explicit and implicit knowledge
   - Use techniques like interviews and ethnographic analysis to do so.
2. **Analyze** and iterate
3. **Specify** for implementation phase
   - SRS and use cases
4. **Validation**: ensure requirements capture customer goals

Design

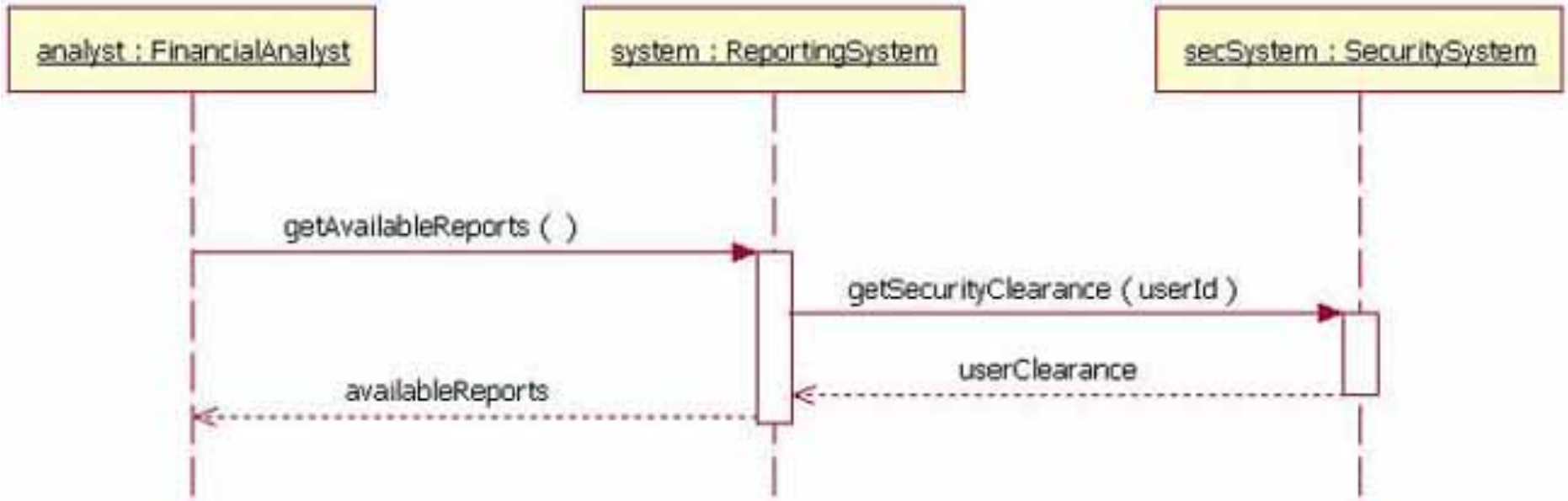
- Design should make programs more maintainable, scalable, etc.
- Design *process* is about communication
- High-level, architectural design
  - Client-server, REST, Peer to peer
- Mid-level, class-level design
  - Class diagrams
- Low-level, object interaction design
  - Sequence diagrams



Class diagrams



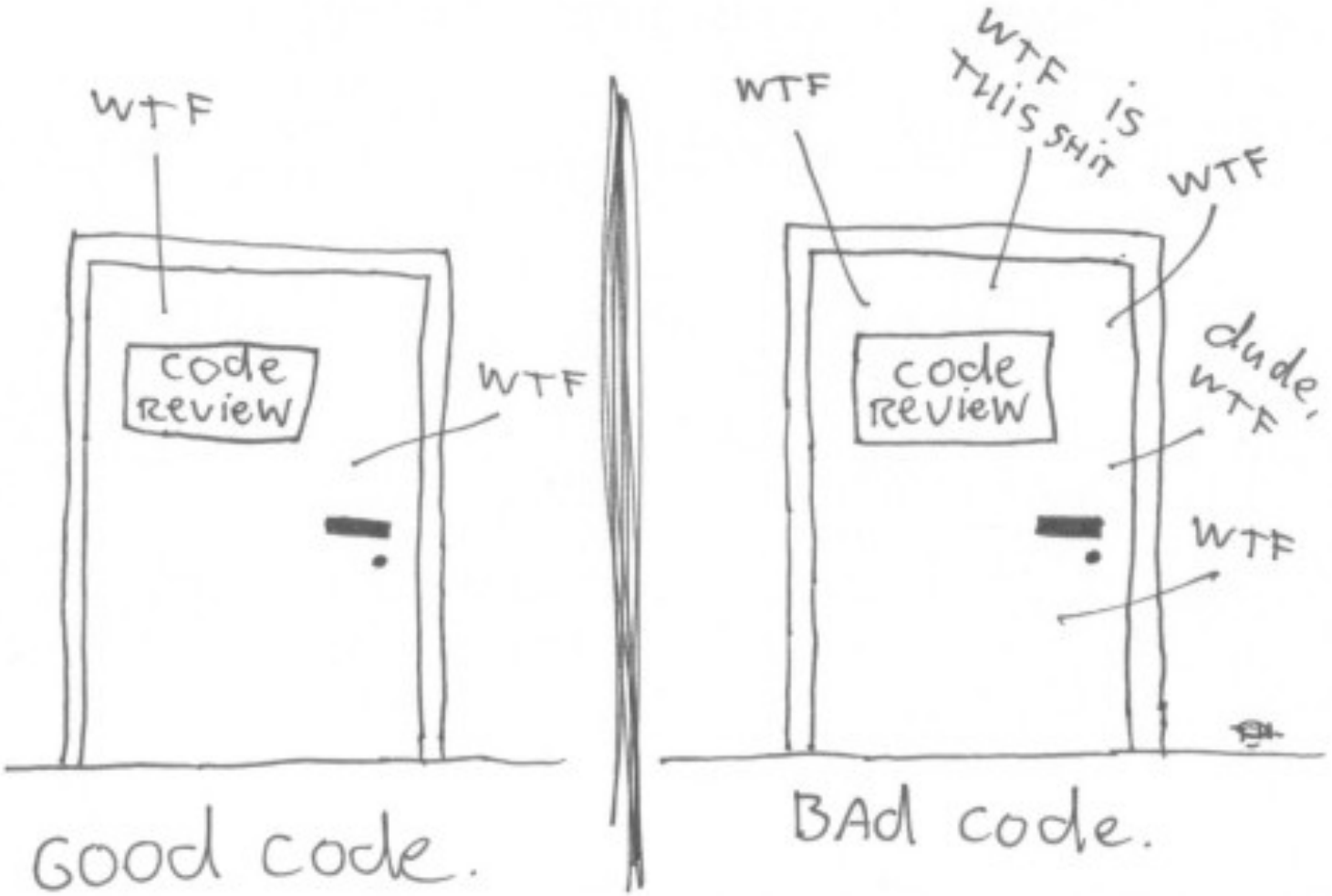
Sequence diagrams



Design principles

- Designs can be better or worse
- Modularity is a good goal (in general)
  - High Cohesion
  - Loose Coupling
  - Information Hiding
  - Open/Closed Principle
  - Liskov Substitution Principle

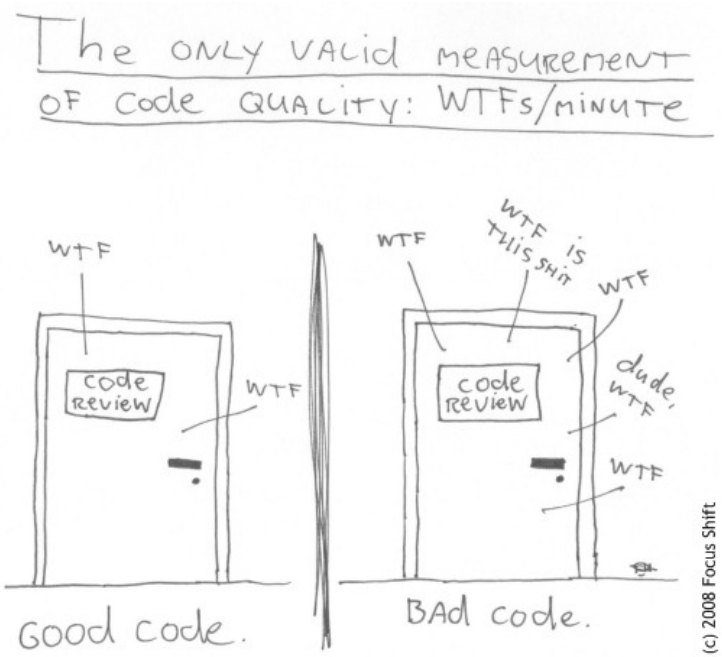
The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE



Which door is yours? How can we end up at the right door?

- (Automated) Analysis
  - Code Metrics
  - Formatting
  - Static and Dynamic Analysis
  - Measuring and Monitoring

- Craftsmanship
  - Code Reviews
  - Clean code
  - Smells and Refactoring



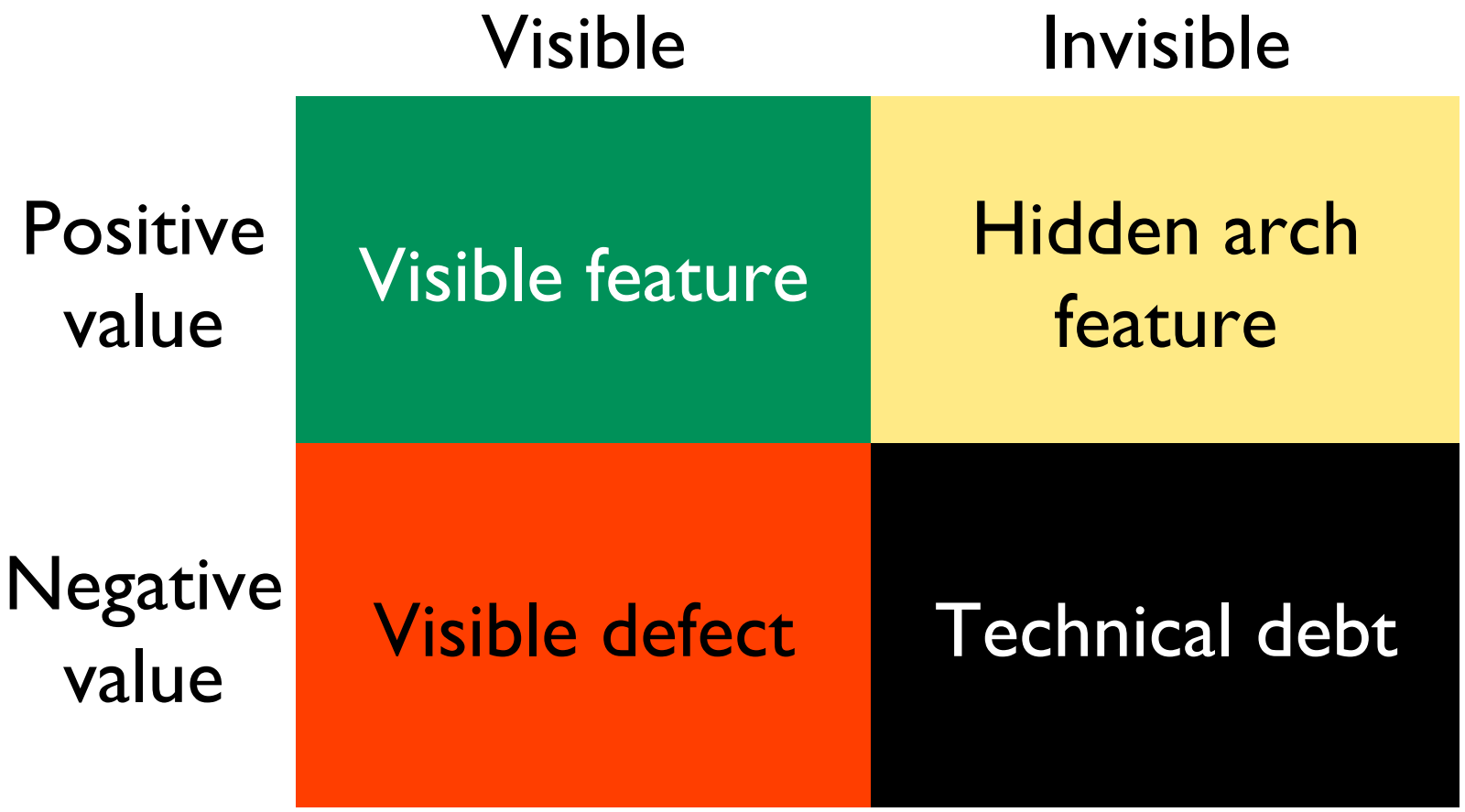
Refactoring and code smells

- Why refactor: evolving software and short iterations.
- Refactoring cycle: tests, code, tests, refactor, tests
- Code smells.
- Refactorings.

From smell to refactoring.

```
public void seek(float posValue) {
    //if fading, ignore
    if (bFading){
        return;
    }
    //save current position
    float fCurrentPos = fPos;
    //Do not seek to a position too near from the end if we are seeking. MAX=98%
    if (posValue>0.98f){
        posValue = 0.98f;
    }
    // leave if already seeking
    if (player != null && getState() == BasicPlayer.SEEKING) {
        Log.warn("Already seeking, leaving"); //$NON-NLS-1$
        return;
    }
    if (mPlayingData.containsKey("audio.type") && player != null) { //$NON-NLS-1$
        Type type = TypeManager.getInstance().getTypeByTechDesc((String) mPlayingData.ge
        // Seek support for MP3. and WAVE
        if (type.getBooleanValue(XML_TYPE_SEEK_SUPPORTED)
                && mPlayingData.containsKey("audio.length.bytes")) { //$NON-NLS-1$ //$NO
            int iAudioLength = ((Integer) mPlayingData.get("audio.length.bytes")).intVal
            long skipBytes = (long) Math.round(iAudioLength * posValue); //$NON-NLS-1$
            try {
                player.seek(skipBytes);
                setVolume(fVolume); //need this because a seek reset volume
            } catch (Exception e) {
                Log.error(e);
                return;
            }
        } else {
            Messages.showErrorMessage("126"); //$NON-NLS-1$
            return;
        }
    }
}
```

Tech debt - invisible and negative value



Testing

- Proving vs. testing
- Types of tests
  - Unit, regression, system, acceptance
- Challenges of testing: too many combinations!
- Test approaches: functional vs structural
- Stopping criteria (coverage?)

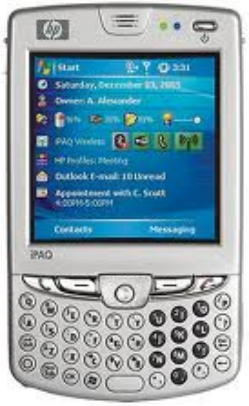
Evolution and maintenance

- Why maintenance is a large part of SE
- Types of software evolution
  - Laws of software evolution!
- Maintenance strategies and evolving APIs.

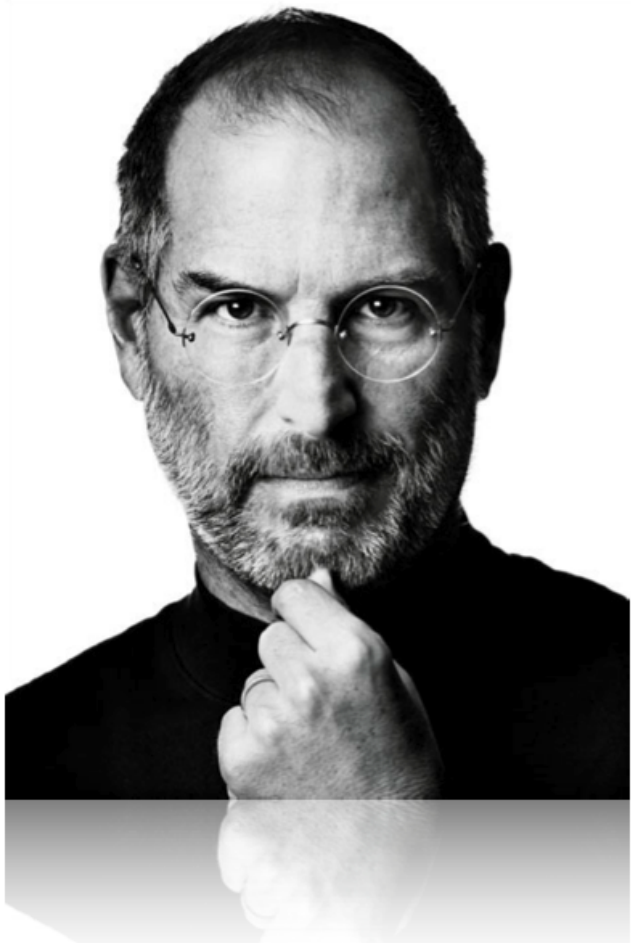
Designing interfaces is hard

How many of you can program or use all aspects of your

- digital watch?
- cell phone?
- DVD player?
- microwave?
- sewing machine?
- washer and dryer?
- stereo system (home or car)?



What is design?



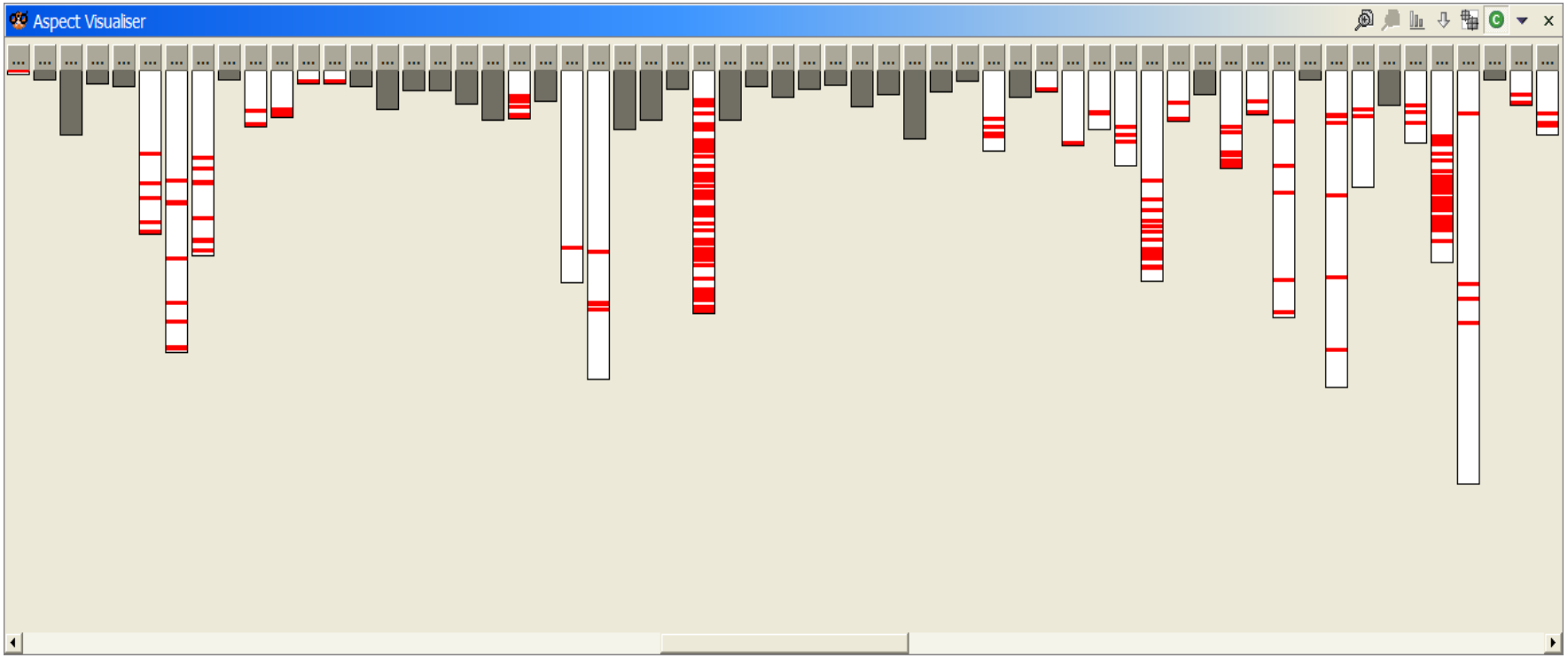
Design is not just what it looks like and feels like.

Design is how it works.

Usability Principles

- Nielsen's 10 Principles of UI Design (Jakob Nielsen)

not so good modularity

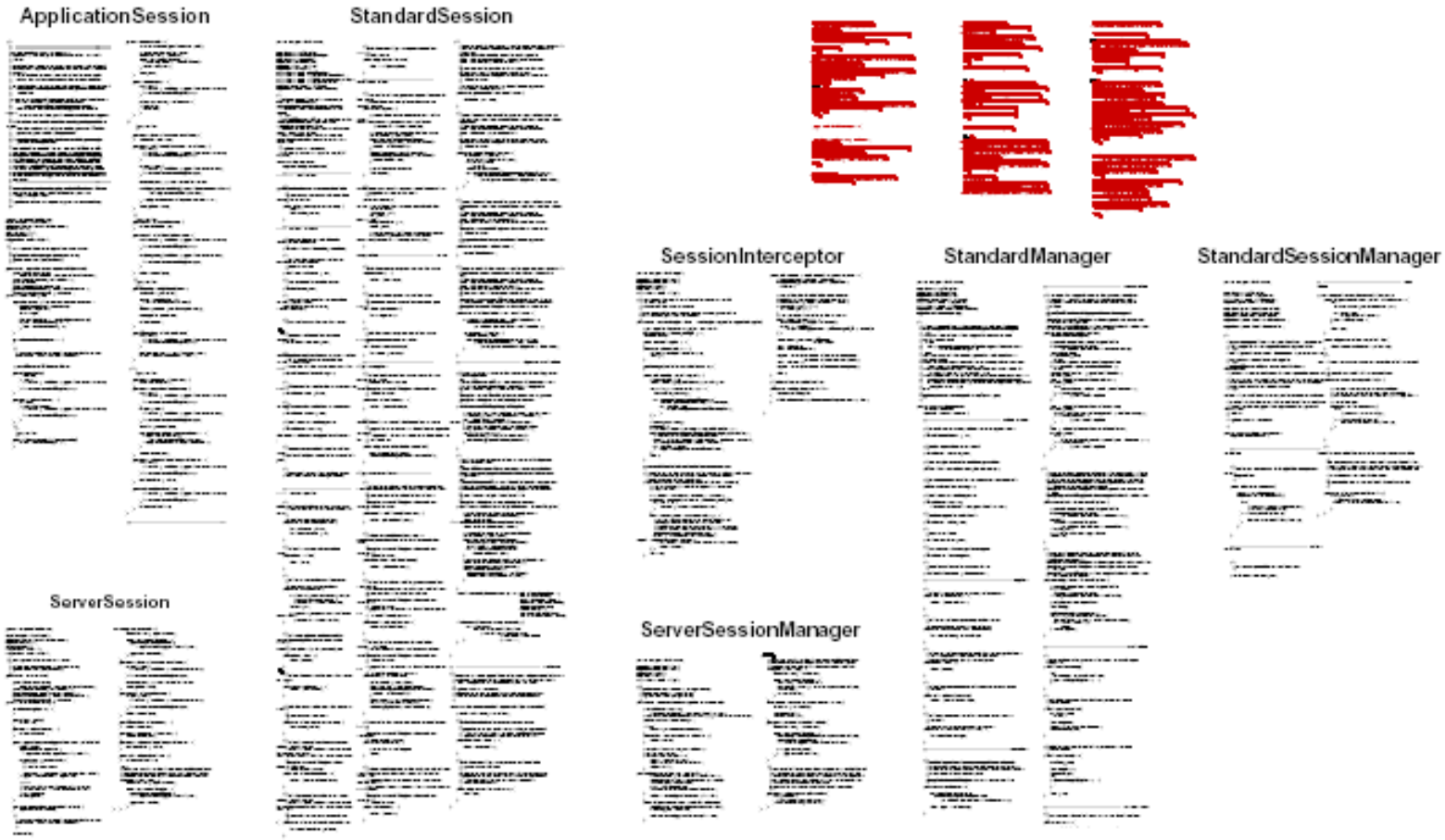


- logging in Tomcat
  - scattered across the packages and classes
  - error handling, security, business rules, …

Without AOP


ApplicationSession
StandardSession
SessionInterceptor
StandardManager
StandardSessionManager
ServerSession
ServerSessionManager

With AOP


ApplicationSession
StandardSession
SessionInterceptor
StandardManager
StandardSessionManager
ServerSession
ServerSessionManager

Be a better software engineer

- Write clean, understandable code not only for yourself, but also for others
- Test your code, measure what you test
- Maintain your design and code.
- Share and be a team player!

Q & A