CS 350 – Software Engineering

November 15, 2017

Week 12, Class #37

Refactoring

Mark Seaman

MWF – 10:00-11:30 - Kepner 0095F

This Week – Risk Managment

Last Week

Dev Ops

Monday, 11-13

Technical Debt

Wednesday, 11-15

Refactoring

Friday, 11-17

Reusable Apps

Next Week

Teams

Refactoring

✧ What is refactoring?

Refactoring

✧ What is refactoring?

- *disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior*
- *The system is kept fully working after each small refactoring, reducing the chances that a system can get seriously broken during the restructuring.*

https://refactoring.com – Martin Fowler

Refactoring

✧ When do you do it?

- Problem code
- Getting ready to add a feature
- After adding a feature

✧ How do you do it?

- Find something that needs improved
- Build a test
- Select a refactoring
- Repeat until done

Refactoring

- ✧ Kent Beck's Rules for Simple Design
  - Runs all tests
  - Contains no duplication
  - Expresses the intent of the programmer
  - Minimizes classes and methods
- ✧ Evolutionary Design
  - Incremental improvements
  - Adapting to new realities

Refactoring Goals

- ✧ Simplicity
- ✧ Testability
- ✧ No duplication
- ✧ Expressive
- ✧ Minimal

Refactorings

- ✧ Rename
- ✧ Move
- ✧ Copy
- ✧ Extract
- ✧ Inline
- ✧ Delete
- ✧ Object
- ✧ Variable
- ✧ Constant
- ✧ Field
- ✧ Method
- ✧ Parameter
- ✧ Superclass

Simpler is better

A good day programming is when you

add 200 lines of code

A great day programming is when you

remove 200 lines of code without breaking anything

Boy scouts and Doctors

Leave the campsite better than you find it.

Do no harm!

Software developers are professionals.

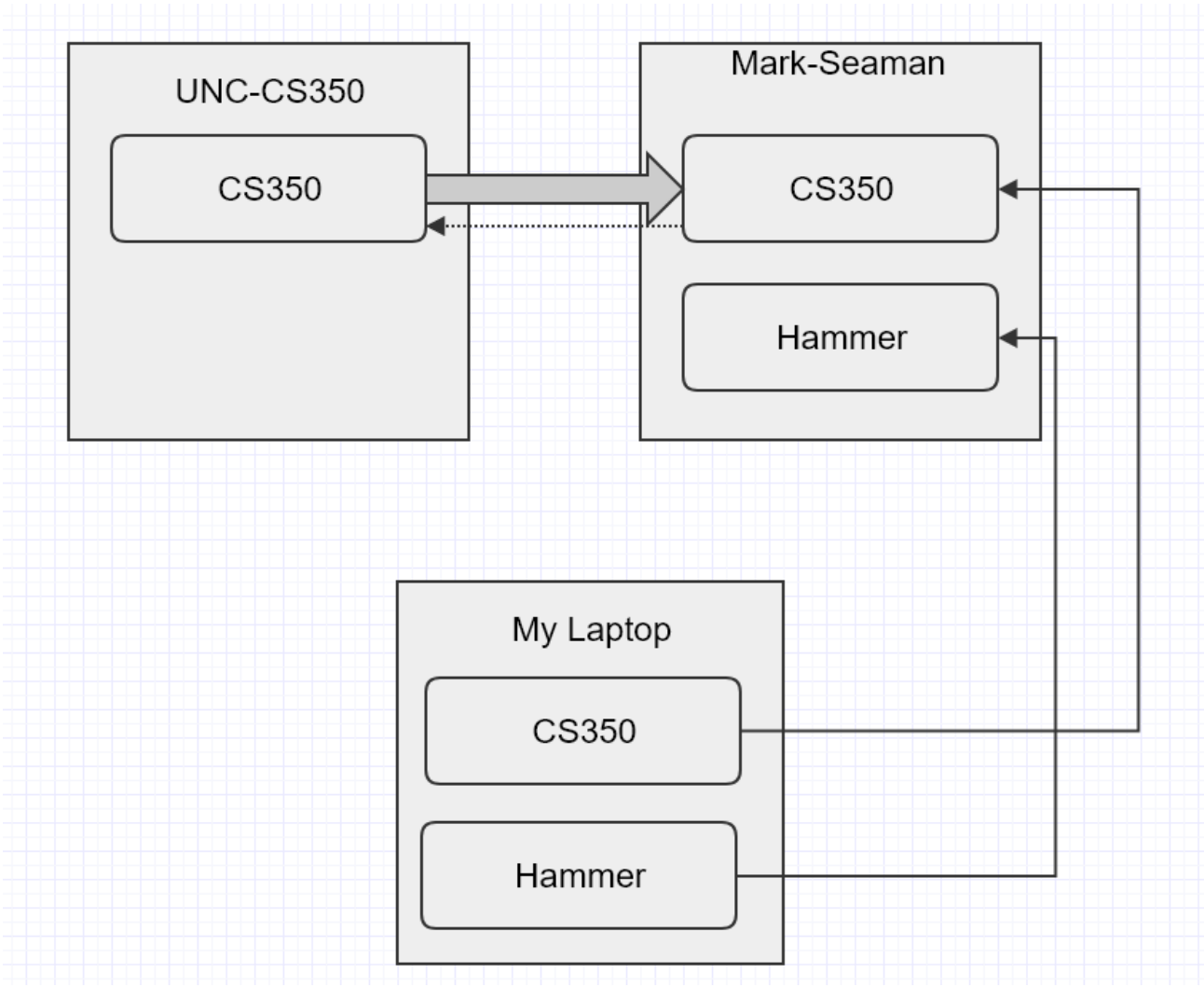
-- Uncle Bob (Robert Martin)

Exercises

- Git Push 11/13 -- Install Github Desktop
- Fork Repo 11/15
- Pull Request 11/27

See you Friday

Github Repos



Technical Debt

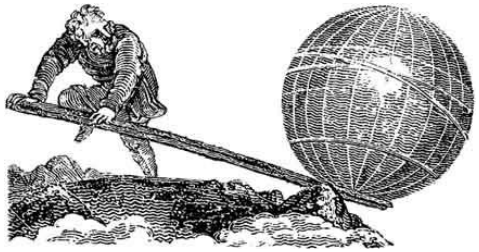
✧ What is technical debt?

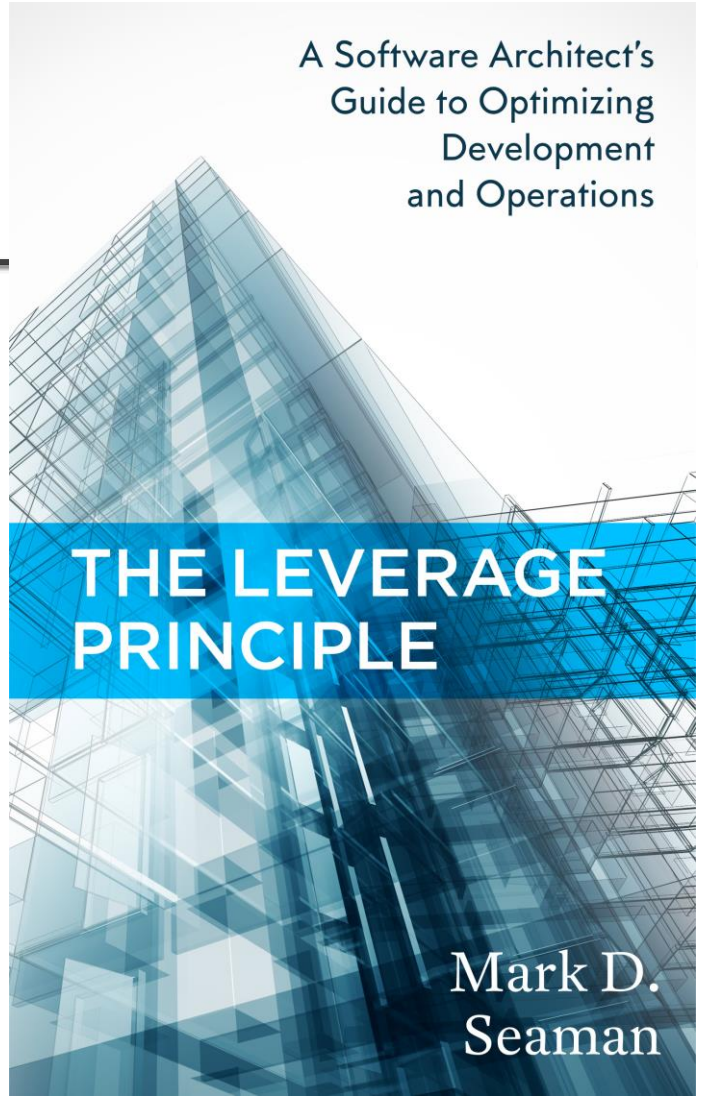
- Promise of later work
- Shortcut for immediate benefit
- Compromise of the right way
- Cumulative effects of bad decisions

Leverage Principle

- ✧ Technical Debt determines cost
- ✧ Best Practices reduce debt

- ✧ Areas of concern
  - Development
  - Operations
  - Teams





Look inside

The Leverage Principle: A Software Architect's Guide to Optimizing Development and Operations Kindle Edition

by Mark D. Seaman (Author), Stacie Seaman (Editor)

3 customer reviews

Software development is expensive and it is far more expensive than it needs to be. The pace of development has increased dramatically with the arrival of cloud-based apps and continuous delivery and the processes for software development and operations have to adapt to this new reality.

Development Lifecycle

- ✧ Technology
  - Fitness of tools used
  - Technical skill level
- ✧ Design
  - Too much or too little design
- ✧ Code
  - Complexity
  - Non-incremental development
- ✧ Test
  - Lack of test or maintenance
  - Poor coverage

Software Operations

- Deployment
  - Capabilities of hosting service
  - Lack of automation
- Release Cycle
  - Too long until next release
- Services
  - Complexity of service interactions
  - Inappropriate scale
- Monitoring
  - Lack of transparency
  - Human observation

Building Teams

- ✧ Knowledge
  - Lacking knowledge management system
- ✧ Teamwork
  - Tolerating bad team members - knowledge hoarders and primadonas
- ✧ Learning
  - Not investing in the training of the developers
- ✧ Project Planning
  - Lack of flexibility
  - Ambiguous priorities
  - No leverage

T

- S
  - x