Agile Software Development

Prof. Dr. Dirk Riehle

Friedrich-Alexander University Erlangen-Nürnberg

AMOS E01

Agenda

1. Software developer
2. Sprint planning
3. Day planning
4. Programming principles
5. Code ownership
6. Technical debt
7. Refactoring

1. Software Developer

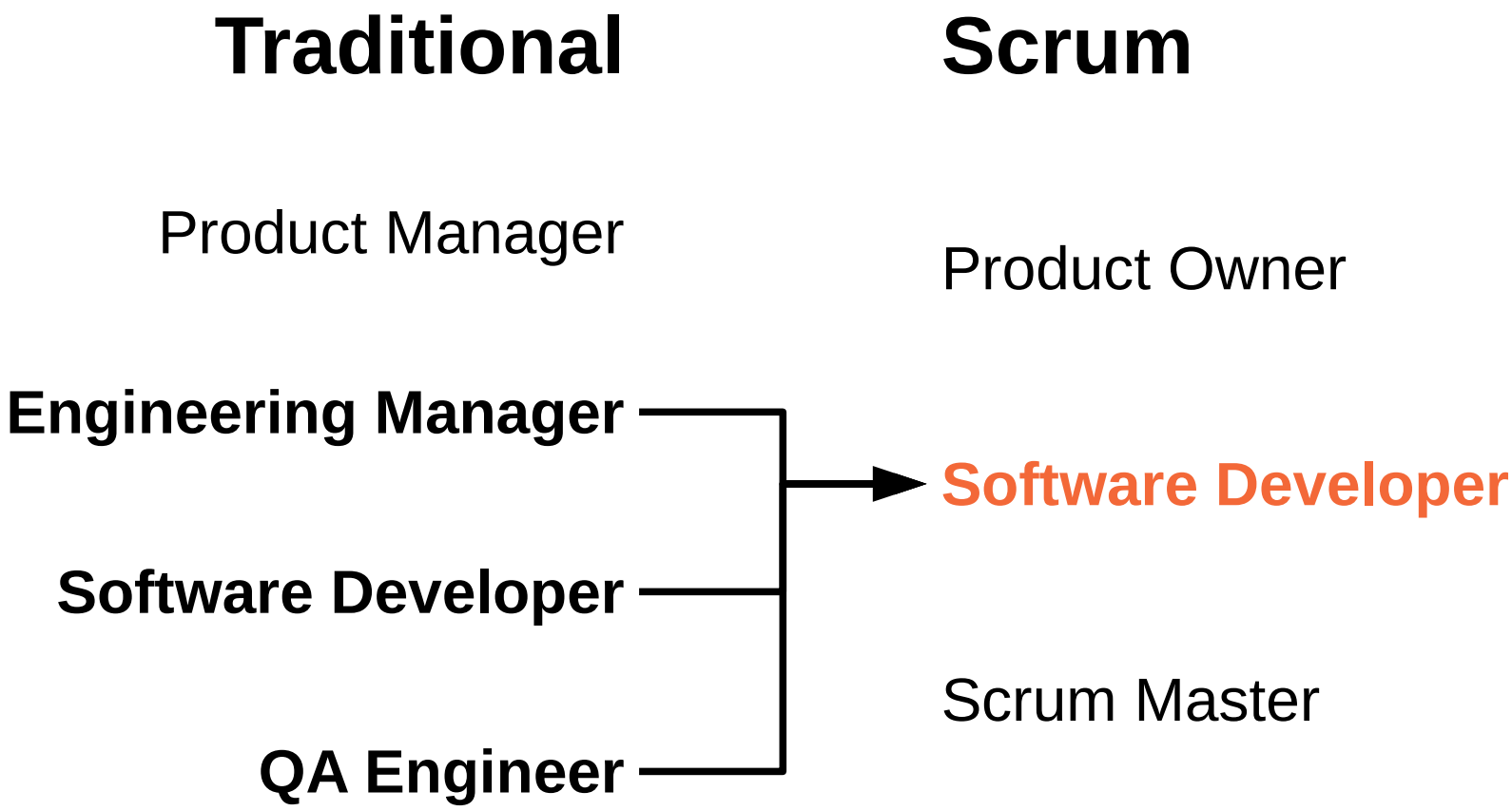
Software Development Team (Recap)

- The **software development team**
  - Holds **overall responsibility** for **delivering working software**
    - That provides the **features** the **team committed to delivering**

Primary Role Responsibilities (Recap)

- Engineering Management
  - Who?
    - By when?
- Software Development
  - How?
    - How fast?
- Quality Assurance
  - Releasable?
    - Good enough?

Traditional to Scrum Role Mapping (Recap)



2. Sprint Planning

Sprint Planning (Practices)

- Break Down Features into Tasks
  - Responsible: Developers
  - Artifacts: Feature, task board, tasks
  - Collaborators: Developers
- Track Feature Implementation
  - Responsible: Developers
  - Artifacts: Sprint backlog, task board
  - Collaborators: Developers

14 Product backlog

- **Show next photo**
  #107 opened by dirkriehle
  Est Size = 8
- **Rate photo and proceed**
  #106 opened by dirkriehle
  Est Size = 8
- **Show or hide photo caption**
  #108 opened by dirkriehle
  Est Size = 8
- **Create default admin**
  #110 opened by dirkriehle
  Est Size = 5
- **Startup and shutdown scripts**
  #122 opened by dirkriehle
  Est Size = 3
- **Reboot and shtudown UI**
  #123 opened by dirkriehle
  Est Size = 5
- **Cold backup and restore**
  #128 opened by dirkriehle
  Est Size = 8
- **Flag photo**
  #124 opened by dirkriehle
  Est Size = 8
- **View flagged photo queue**
  #125 opened by dirkriehle
  Est Size = 8
- **Review flagged photo**
  #126 opened by dirkriehle
  Est Size = 5
- **Track repeat offender**
  #127 opened by dirkriehle

1 Sprint backlog

- **Delete photo**
  #121 opened by dirkriehle
  Est Size = 5

2 In progress

- **Select photo**
  #119 opened by dirkriehle
  Est Size = 5
- **Change photo data**
  #120 opened by dirkriehle
  Est Size = 3

1 Awaiting review

- **Browse photo portfolio**
  #109 opened by dirkriehle
  Est Size = 8 Real Size = 8

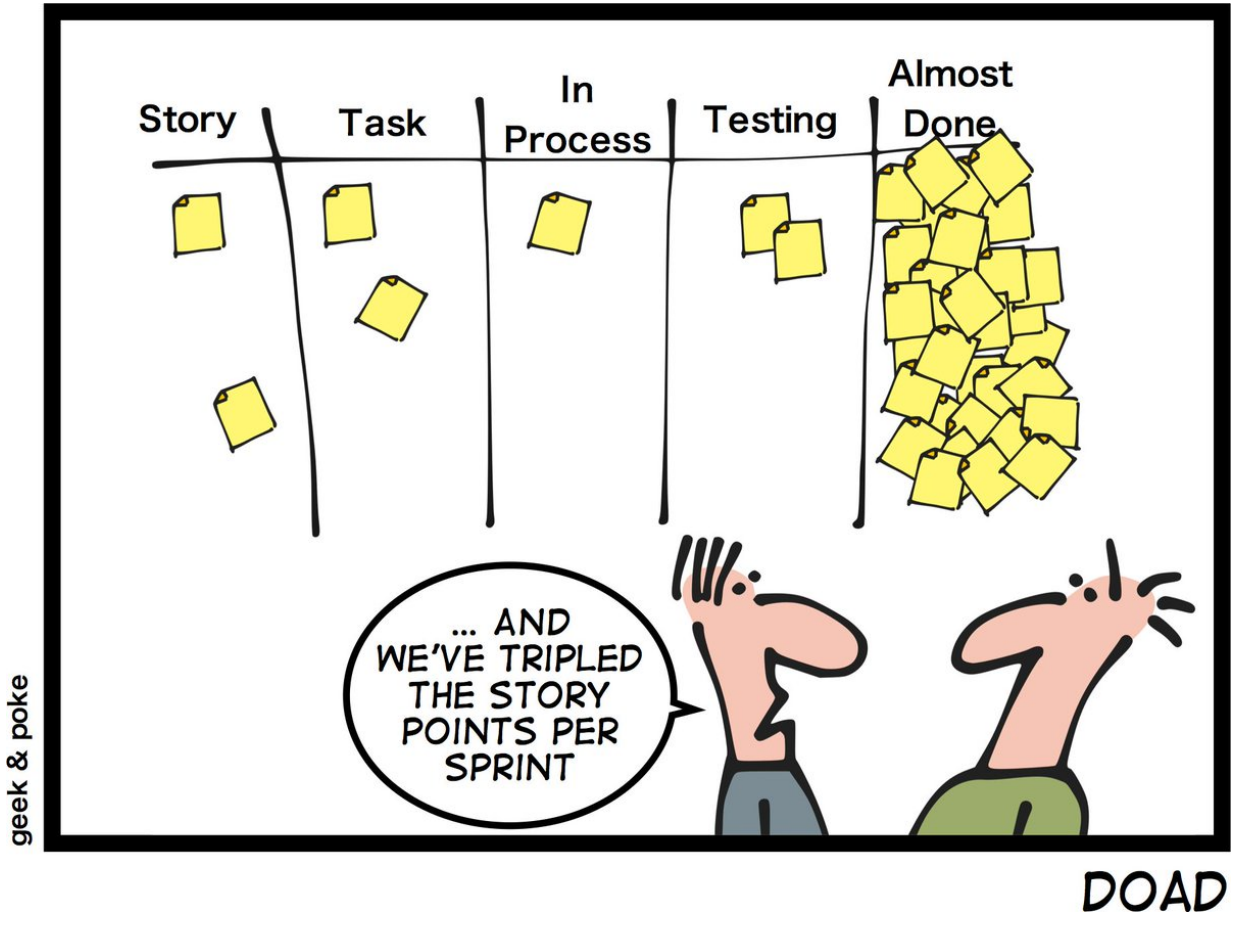
8 Feature archive

- **Upload photo**
  #118 opened by dirkriehle
  Est Size = 8 Real Size = 8
- **Change password**
  #117 opened by dirkriehle
  Est Size = 3 Real Size = 5
- **Change basic profile**
  #116 opened by dirkriehle
  Est Size = 5 Real Size = 5
- **Prompt basic profile**
  #115 opened by dirkriehle
  Est Size = 5 Real Size = 5
- **Reset password**
  #114 opened by dirkriehle
  Est Size = 5 Real Size = 5
- **Logout**
  #113 opened by dirkriehle
  Est Size = 3 Real Size = 3
- **Login**
  #112 opened by dirkriehle
  Est Size = 5 Real Size = 5
- **Register**
  #111 opened by dirkriehle
  Est Size = 8 Real Size = 8

Task Board (Artifact)

- Task board (a.k.a. Kanban board)
  - Visualizes the progress towards finishing features and the current sprint
  - Shows for each feature the progress of implementation
  - May break down work into hours

Definition of Almost Done



3. Day Planning

Day Planning (Practices)

- Perform Daily Scrum
  - Responsible: Scrum master
  - Artifacts: Impediment backlog
  - Collaborators: Developers

Daily Scrum

- Daily Scrum
  - Is a daily status meeting to sync on problems and upcoming work
  - Is to be kept as short as possible (a.k.a. daily stand-up meeting)
- Other properties
  - “Pigs” are mandatory, “chicken” are optional, only “pigs” may speak
  - Provides only updates, everyone may only speak once
  - No discussions allowed, any discussions are taken off-line
  - Scrum master to follow-up on problems
- Questions asked
  - What did you do yesterday?
  - What will you be doing today?
  - What obstacles are in your way?

Plank Meetings

4. Programming Principles

General Principles 1 / 4

KISS

(Keep It Simple, Silly)

General Principles 2 / 4

YAGNI

(You Ain’t Gonna Need It)

General Principles 3 / 4

DRY

(Don’t Repeat Yourself)

General Principles 4 / 4

1. Make it run
2. Make it right
3. Make it fast

5. Code Ownership

Collective Code Ownership (Practice)

- Own code collectively
  - Responsible: Development team
  - Artifacts: Source code
  - Collaborators: Developers

Collective Code Ownership

- Definition and purpose
  - Provide full read and write access to each developer of the collective
  - Is to instill a feeling of overall responsibility for the code base
  - The opposite of collective code ownership is individual code ownership
- Other properties
  - Every member of the collective
    - Cares about the architecture
    - Cares about clean code
    - Ensures high quality
  - Every single author can
    - Implement a feature end-to-end
    - Finish a refactoring
    - Fix a bug

Programming Standard (Practice)

- Use programming standard
  - Responsible: Development team
  - Artifacts: Programming standard, source code
  - Collaborators: Developers

Programming Standard

- Definition and purpose
  - Is a set of rules and conventions that determines naming, formating and structuring of source code and related artifacts
  - Is used to ensure that every developer can read and modify every other developer's code as easily as possible
- Other properties
  - Ease reading source code
  - Ease navigating code
  - Should be mandatory

Original Java Coding Guidelines

3 - File Organization

A file consists of sections that should be separated by blank lines and an optional comment identifying each section.

Files longer than 2000 lines are cumbersome and should be avoided.

For an example of a Java program properly formatted, see "Java Source File Example" on page 19.

3.1 Java Source Files

Each Java source file contains a single public class or interface. When private classes and interfaces are associated with a public class, you can put them in the same source file as the public class. The public class should be the first class or interface in the file.

Java source files have the following ordering:

- Beginning comments (see "Beginning Comments" on page 4)
- Package and Import statements
- Class and interface declarations (see "Class and Interface Declarations" on page 4)

3.1.1 Beginning Comments

All source files should begin with a c-style comment that lists the class name, version information, date, and copyright notice:

```
/*
 * Classname
 *
 * Version information
 *
 * Date
 *
 * Copyright notice
 */
```

Copy

3.1.2 Package and Import Statements

Categories and Examples of Method Types

Query Method	Mutation Method	Helper Method
get method (getter)	set method (setter)	factory method
boolean query method	command method	cloning method
comparison method	initialization method	assertion method
conversion method	finalization method	logging method
...	...	...

Conventions / Patterns Beyond Formatting

- Naming conventions
  - Attributes, methods
  - Classes, packages
  - ...
- Design conventions
  - Collaborations
  - Modules, classes
  - ...
- Package structures
  - …
- **See our Advanced Design and Programming (ADAP) course**

Quiz: Software Development

1. A file may have many authors. Should the names of these authors be listed in the file's header?
   - Yes
   - No

6. Technical Debt

Definition of Big Ball of Mud [FY97]

A Big Ball of Mud is a haphazardly structured, sprawling, sloppy, duct-tape-and-baling-wire, spaghetti-code jungle. These systems show unmistakable signs of unregulated growth, and repeated, expedient repair. Information is shared promiscuously among distant elements of the system, often to the point where nearly all the important information becomes global or duplicated. The overall structure of the system may never have been well defined. If it was, it may have eroded beyond recognition. Programmers with a shred of architectural sensibility shun these quagmires. Only those who are unconcerned about architecture, and, perhaps, are comfortable with the inertia of the day-to-day chore of patching the holes in these failing dikes, are content to work on such systems.

Benefits of Good (“Well-Factored”) Code

1. **Maintainability**
   (Easier to understand)
2. **Extensibility**
   (Easier to adapt and evolve)
3. **Predictability**
   (Improves planning ability)

Video on Technical Debt [1]

Technical Debt

(Ward Cunningham)

[1] See https://youtu.be/Jp5japiHAs4

Video Lessons

- Technical debt is a metaphor
  - Communicates well to manager
  - (Certainly in financial services)
- Taking on debt can speed up development
  - It may be justified to learn faster
  - But you have to pay up later
- If you don’t pay back debt you’ll slow down
  - Paying up means refactoring
  - Still, never write poor code deliberately

Handling Technical Debt

1. **Identify problem**
   (So-called “code smells”)
2. **Identify need to act**
   (Correlate occurrences)
3. **Know how to act**
   (Refactor code)

Code Smell

- According to Fowler [F99]
  - “smells are certain structures in the code that indicate violation of fundamental design principles and negatively impact design quality”
  - also, “a code smell is a surface indication that usually corresponds to a deeper problem in the system”
- Code smells are not bugs

Example Code Smells

- Duplicated code
- Long method
- Large class
- ...

7. Refactoring

Refactoring (Practice)

- Definition and purpose
  - Is a behavior-preserving transformation of existing source code
  - It is a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior
- More on refactoring
  - Change the structure of code without changing behavior
  - Focus on non-functional features within range of specification
  - Are defined techniques that are typically language specific
  - Are ideally supported by IDEs, for example, the Eclipse JDT
- Defined by Opdyke [O92], popularized by Fowler [F99]

Example Refactorings

- Rename class
- Pull-up field
- Extract method
- …

- More at https://refactoring.com

Example Extract-Method Refactoring 1 / 2

```
public class PhotoManager extends ObjectManager {
     protected Map<PhotoId, Photo> allPhotos = new HashMap<PhotoId, Photo>();

     public void addPhoto(Photo photo) {
          PhotoId id = photo.getId();
          assertIsNewPhoto(id);
          allPhotos.put(id, photo);
          ...
     }

     public Photo getPhotoFromId(PhotoId id) {
          Photo result = doGetPhotoFromId(id);
          if (result == null) {
               ...
               if (result != null) { allPhotos.put(id, result); }
          }
          return result;
     }

     public Set<Photo> findPhotosByOwner(String ownerName) {
          ...
          for (Iterator<Photo> i = r.iterator(); i.hasNext();) {
               Photo photo = i.next();
               allPhotos.put(photo.getId(), photo);
          }
          return r;
     }
     ...
```

Example Extract-Method Refactoring 2 / 2

```
public class PhotoManager extends ObjectManager {

     public void addPhoto(Photo photo) {
          ...
          doAddPhoto(photo);
          ...
     }

     public Photo getPhotoFromId(PhotoId id) {
          ...
          doAddPhoto(photo);
          ...
     }

     public Set<Photo> findPhotosByOwner(String ownerName) {
          ...
          for (Iterator<Photo> i=r.iterator(); i.hasNext(); ) {
               doAddPhoto(i.next());
          }
          ...
     }

     protected void doAddPhoto(Photo myPhoto) {
          allPhotos.put(myPhoto.getId(), myPhoto);
     }

     ...
```

When to Refactor?

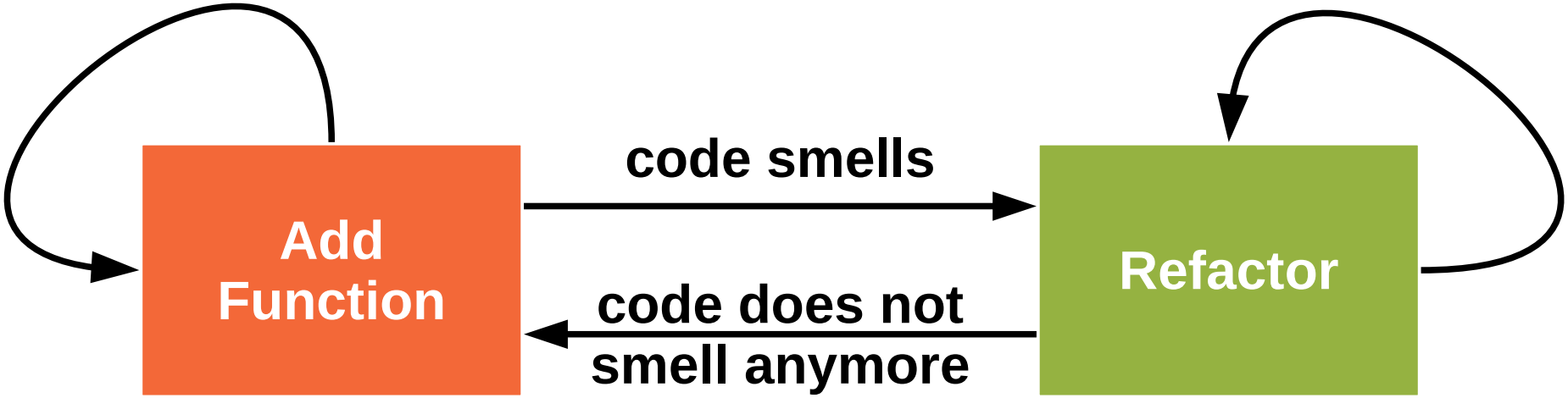
The “three strikes” rule

1st time: Just do it

2nd time: Wince at duplication

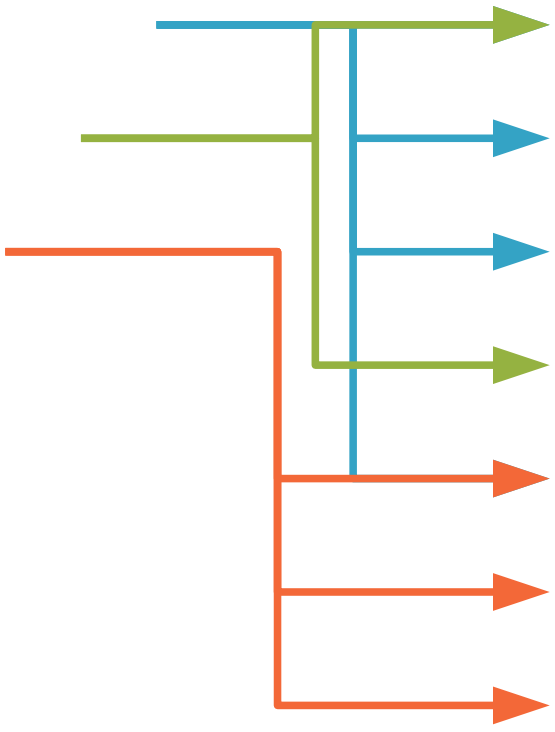
3rd time: Refactor

Refactoring Process (Two Hats)



Code Smells and Refactorings

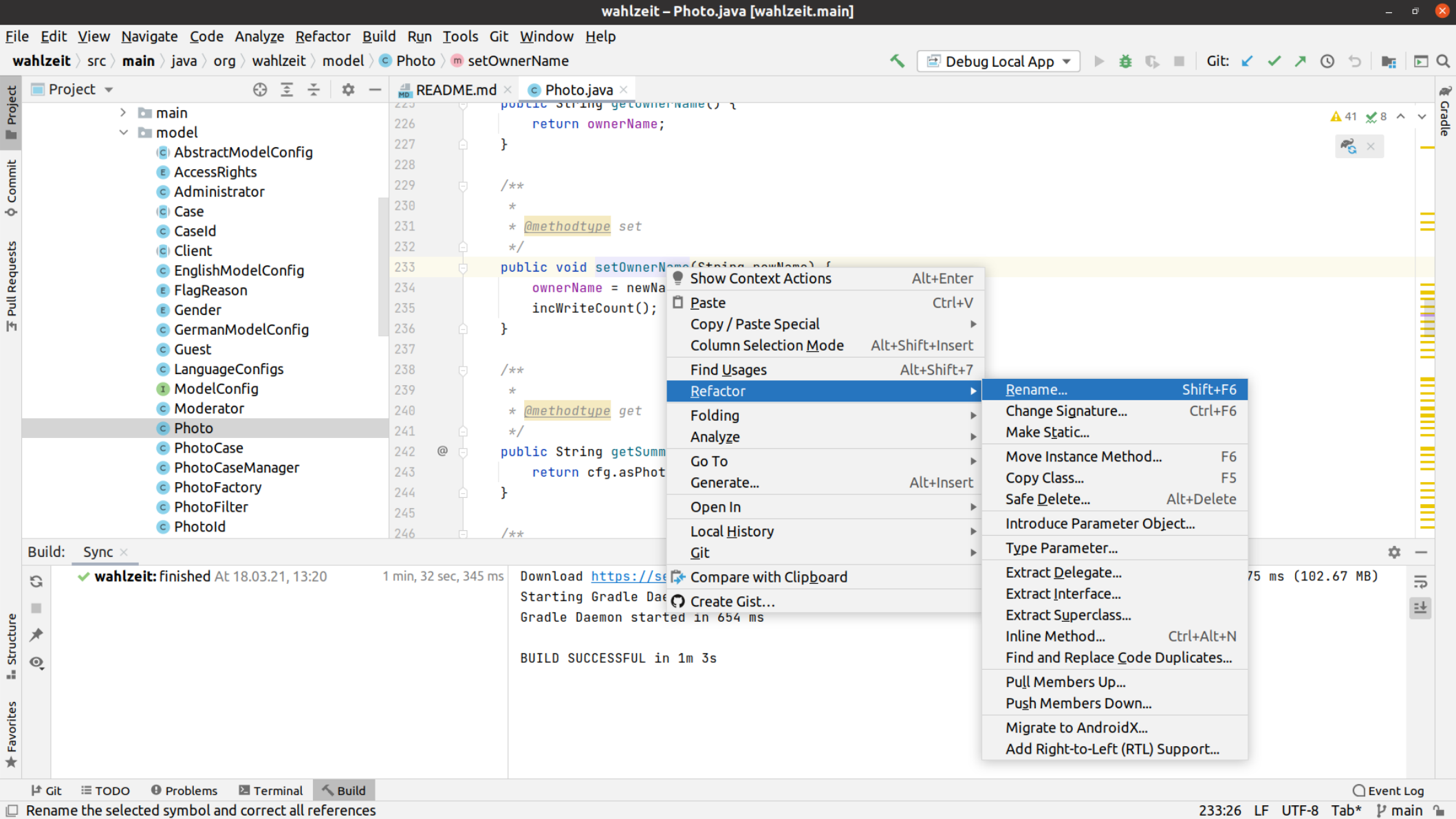
- Duplicated code
- Long method
- Large class
- ...



- Extract method
- Pull up field
- Form template method
- Decompose conditional
- Extract class
- Extract subclass
- Extract interface
- ...

Example Refactoring Process

- The refactoring process can become complex
  - It may turn into a series of refactorings
- Example removal of switch statement
  - Extract method
  - Move method
  - Replace type code …
    - with subclass or
    - with state / strategy object
  - Replace conditional ...
    - with polymorphism


wahlzeit – Photo.java [wahlzeit.main]
File Edit View Navigate Code Analyze Refactor Build Run Tools Git Window Help
wahlzeit › src › main › java › org › wahlzeit › model › Photo › setOwnerName
Debug Local App
Git:
Project
README.md
Photo.java
main
model
AbstractModelConfig
AccessRights
Administrator
Case
CaseId
Client
EnglishModelConfig
FlagReason
Gender
GermanModelConfig
Guest
LanguageConfigs
ModelConfig
Moderator
Photo
PhotoCase
PhotoCaseManager
PhotoFactory
PhotoFilter
PhotoId
226 return ownerName;
227 }
229 /**
230 *
231 * @methodtype set
232 */
233 public void setOwnerName(String newName) {
234 ownerName = newNa
235 incWriteCount();
236 }
238 /**
239 *
240 * @methodtype get
241 */
242 public String getSumm
243 return cfg.asPhot
244 }
246 /**
Show Context Actions Alt+Enter
Paste Ctrl+V
Copy / Paste Special
Column Selection Mode Alt+Shift+Insert
Find Usages Alt+Shift+7
Refactor
Folding
Analyze
Go To
Generate... Alt+Insert
Open In
Local History
Git
Compare with Clipboard
Create Gist...
Rename... Shift+F6
Change Signature... Ctrl+F6
Make Static...
Move Instance Method... F6
Copy Class... F5
Safe Delete... Alt+Delete
Introduce Parameter Object...
Type Parameter...
Extract Delegate...
Extract Interface...
Extract Superclass...
Inline Method... Ctrl+Alt+N
Find and Replace Code Duplicates...
Pull Members Up...
Push Members Down...
Migrate to AndroidX...
Add Right-to-Left (RTL) Support...
Build: Sync
wahlzeit: finished At 18.03.21, 13:20 1 min, 32 sec, 345 ms
Download https://se
Starting Gradle Da
Gradle Daemon started in 654 ms
BUILD SUCCESSFUL in 1m 3s
75 ms (102.67 MB)
Git TODO Problems Terminal Build
Event Log
Rename the selected symbol and correct all references
233:26 LF UTF-8 Tab* main

Quiz on Refactoring

- Your code smells. All signs for refactoring are given. Under which circumstances should you not start a refactoring?

Summary

1. Software developer
2. Sprint planning
3. Day planning
4. Programming principles
5. Code ownership
6. Technical debt
7. Refactoring

Thank you! Questions?

dirk.riehle@fau.de – https://oss.cs.fau.de

dirk@riehle.org – https://dirkriehle.com – @dirkriehle

Legal Notices

- License
  - Licensed under the CC BY 4.0 International license
- Copyright
  - © 2010-2021 Dirk Riehle, some rights reserved