Introduction to Agile Software Engineering in SAFe
Harry Koehnemann
SAFe Consultant and Fellow
Scaled Agile, Inc
Why the need for Agile Software Engineering?

- Difficult to understand and change system
- Late discovery of errors in integration and in production
- Missed release dates
- Difficulty releasing due to many architectural dependencies
- Changes in one part of system introduces errors in another
Agile Software Engineering focuses on quality

- Lean and Agile principles and practices
- Behavior-Driven Development (BDD)
- eXtreme Programming (XP)
- Code quality
- Design patterns and practices
- Agile modeling
Elements of software engineering

- Thinking test-first
- Support the continuous flow of value
- Aligning on what to build with Behavior-Driven Development (BDD)
- Build quality in with design and code quality
Thinking test-first
Traditional testing (V-Model) delays feedback
Shift testing left for fast and continuous feedback

Determine Feature → Test Feature → ... always testing...

Write Story → Test Story → ... always testing...

Write Code → Test Code → ... always testing...
Write testable features, stories, and code

Feature: Cruise control with speed limit finder

Benefit hypothesis:
Drivers would like cruise control to obey speed limits

Measurement:
1. 30% of drivers will use the speed limit finder when on cruise control

As a ... I want ... So that ...
Acceptance Test:
Given a speed limit of 50 mph
When the car drives
Then the car speed should be between 49 and 50 mph

```java
@Test
public void MapperShouldReturnSpeedLimit() {
    Location loc = new Location(Lat(50.12), Lon(40.34))
    Speed speed = Mapper.getSpeedLimit(loc)
    assertEquals(new Speed(50), speed)
}
```
Early, automated tests build a balanced test portfolio

- Test Pyramid advocates many small, low-level, automate tests and fewer large, manual tests.

![Test Pyramid Diagram]

- Traditional testing (Find defects):
  - Large (Slow)
  - Medium
  - Small (Fast)

- Agile testing (Prevent defects):
  - UI
  - Medium
  - Small (Fast)

**End-to-end UI tests**

**External services**

**Individual classes**
Assess your organization’s testing practices

- Organize into groups and consider the following questions about your organization’s testing process
  - How closely are tests defined and executed relative to the requirement being specified?
  - What is your testing balance between low-level, automated, fast tests and high-level, manual, slow tests?
  - Do either of these cause a problem at your organization?
Support the continuous flow of value
Agile Software Engineering intersects DevOps

Agile Software Engineering focus:

- Write good, testable stories (BDD)
- Develop small change with tests (TDD)
- Build/test small change
- Validate in prod-simulated env
- Validate on prod-grade env
- Deploy for feedback
Support continuous testing through the Pipeline

Development

System

Staging

Production

Dev smoke tests

QA tests (scaleability, security, UAT)

Environment tests

Artifact repository
Aligning on what to build with Behavior-Driven Development (BDD)
Understanding behavior involves multiple perspectives

Cognitive diversity in thinking perspectives:

- **Customer thinking:**
  - Viable, desirable

- **Developer thinking:**
  - Technically feasible

- **Tester thinking:**
  - Exceptions, edge case, boundary conditions

THE TRIAD
Represent unambiguous behavior with BDD

- Behavior is often first described in general terms, which can be ambiguous
- Specific examples of behavior can lead to more understanding

**Given** a speed limit of 50 mph
**When** the car drives
**Then** the car speed should be between 49 and 50 mph

What are some other specific tests?

As a driver, I want the vehicle to determine the speed limit and set the speed to that limit
So that I do not have to pay attention to speed limits

Acceptance Criteria:
- Car maintains speed close to speed limit but not above it
Build quality in with design and code quality
Build quality into code and designs

Continuous attention to technical excellence and good design enhances agility
– Agile Manifesto

- Investing in quality helps avoid rework and defect cost of delays

- Code quality
  - Effective coupling and cohesion**
  - Good abstraction and encapsulation**
  - Collective ownership
  - Refactoring
  - TDD

- Design Quality
  - Intentional programming**
  - Applying SOLID Principles
  - Applying Design Patterns

** discussed in this presentation
Create highly cohesive, loosely coupled entities

- Responsibilities are closely related to common purpose
- Guides how we decompose components, classes, methods, etc.
- Enables flow by supporting independently deployable units
Types of Coupling

- **Loose** - Entities communicate through well-defined interface
- **Tight** – Aware of other entity’s implementation

```java
class SpeedController {
    myEngine.throttleValue = 12
}
```

```java
class SpeedController {
    myEngine.accelerate()
}
```
Abstraction / Encapsulation

- **Abstraction:**
  - Use domain terms that are not dependent on implementation

- **Encapsulation:**
  - Hide implementation

- Provides *seams* to inject test doubles

How long does it take you to learn to drive a new car? Turn on the lights? Tune the radio?
Use separation to decompose system elements

- Programming by intention:
  - Separate sequencing of operations from individual operations
  - Separate policy (what to do) from implementation (how to do it)

- Separation of concerns:
  - Split responsibilities into separate entities with high cohesion
  - Separate, separate, separate
    - It is easier to lump a splitter than to split a lumper
**Separation example: Database access**

*Programming by intention* locally separates storage logic (retrieveRoute, storeRoute)

*Separation of concerns* delegates retrieval, storage, and other access to another object, DBConnection

```java
class RouteManagement {
    addLocationToRoute(RouteID id, Location location) {
        Route route = retrieveRoute(id)
        route.add(location)
        storeRoute(route)
    }

    Route retrieveRoute(RouteID id) {
        DBConnection conn = // DB access object
        // use connection for retrieval
    }
    storeRoute(Route route) {
        DBConnection conn = // DB access object
        // use connection for storage
    }
}
```
Separation also creates *seams* for test doubles

Test doubles extend to provide alternative storage and retrieval logic for testing

```java
class RouteManagementTestDouble extends RouteManagement {
    Route retrieveRoute(RouteID id) {
        assert(....)
        return new Route(....)
    }
    storeRoute(Route route) {
        assert(....)
    }
}
```
Assess your organization’s code and design quality

- Organize into groups and consider the following questions about your organization’s testing process
  - What examples of poor quality exist in your system today?
  - Could improving the quality make the system more modifiable? Testable?
Building in quality supports emergent design

- Design emerges as new requirements are added
  - Do not create capabilities that are not needed for current requirements

- Tradeoff between:
  - Design specifically for current requirement
  - Design generically for next requirement

- Refactor to add new requirements
  - High quality code easier to refactor for design changes
Test-driven development

BDD

Continue until external acceptance test passes

Pick an external acceptance test

Create test that represents the design

Do the design

Internal Implementation Tests (TDD)

TDD

As a ...
I want ...
So that ...
Acceptance Test:
Given a speed limit of 50 mph
When the car drives
Then the car speed should be between 49 and 50 mph
The Test-Driven Development (TDD) cycle

1. Write the test
2. Check if test fails
3. Write code to pass the test
4. Refactor code if necessary
5. Check if all other tests pass
Video: TDD in Action - Prime Factors (Java)

https://youtu.be/3WWv3QBhr9k

```java
class PrimeFactors {
}
```

```java
class PrimeFactorsTest {
}
```
Summary
Agile Software Engineering builds quality in

- UI Tests
- Automated acceptance tests (BDD)
- Automated Unit tests (TDD)

Intentional architecture
Emergent design
Code quality

Architect
Product Owner

Lean UX
BDD
TDD

Agile team