

Table of Contents

Privacy Notice	1
Lesson 1: Introducing the Course	5
1.1 Define Agile software engineering	8
1.2 Form your Agile software engineering team	10
Lesson 2: Connecting Principles and Practices to Built-In Quality	17
2.1 Connect principles to practices	19
2.2 Describe XP practices	22
2.3 Explore key definitions	27
2.4 Describe Built-In Quality	30
2.5 Describe tradeoffs	34
Lesson 3: Accelerating Flow	41
3.1 Outline the flow of value	43
3.2 Validate the benefit hypothesis	47
Lesson 4: Applying Intentional Architecture	59
4.1 Apply systems thinking	60
4.2 Explain the role of Agile architecture in supporting Lean-Agile development	63
4.3 Architect and design for testability	69
Lesson 5: Thinking Test-First	79
5.1 Shift testing left	81
5.2 Explain the Agile testing matrix	85
5.3 Describe the role of nonfunctional requirements (NFRs)	94
5.4 Build quality in throughout the pipeline	99

Lesson 6: Discovering Story Details.....	109
6.1 Examine Story criteria	111
6.2 Split stories to reduce the minimum marketable feature (MMF)	122
6.3 Create workflow or story maps.....	128
6.4 Identify assumptions and risks	132
Lesson 7: Creating a Shared Understanding with Behavior- Driven Development (BDD)	139
7.1 Apply Behavior-Driven Development (BDD) for shared understanding	141
7.2 Specify desired behavior for domain terms	148
7.3 Describe behavior for business rules and algorithms with tests	151
7.4 Test the user interface (UI)	158
7.5 Apply test doubles to BDD	161
7.6 Find existing tests impacted by new requirements.....	165
Lesson 8: Communicating with Models	173
8.1 Explain the use of models	175
8.2 Outline static models.....	179
8.3 Demonstrate class-responsibility-collaboration (CRC).....	186
8.4 Outline dynamic models.....	191
8.5 Outline state models	194
Lesson 9: Building Systems with Code Quality	203
9.1 Identify code qualities.....	205
9.2 Describe cohesion and coupling	209
9.3 Describe other code qualities	216
9.4 Explain the benefits of collective ownership	228

Lesson 10: Building Systems with Design Quality	233
10.1 Explore design tradeoffs	235
10.2 Explain Interface-Oriented Design	238
10.3 Apply quality decomposition practices	246
10.4 Apply differentiation and synthesis	254
10.5 Apply software design patterns	257
Lesson 11: Implementing with Quality	273
11.1 Design with tests	275
11.2 Apply test-driven development (TDD) practices	278
11.3 Implement test doubles and test data	284
11.4 Refactor to support new behavior of the code	289
11.5 Practice emergent design	299
Lesson 12: Course Review	309
12.1 Summarize Agile software engineering	310
12.2 Review your action plan for adopting ASE principles and practices	315
Lesson 13: Becoming a Certified SAFe Professional	319
13.1 Becoming a Certified SAFe Professional	320

Privacy Notice

Your name, company, and email address will be shared with Scaled Agile, Inc. for course fulfillment, including testing and certification. Your information will be used in accordance with the Scaled Agile privacy policy available at <https://www.scaledagile.com/privacy-policy/>.

Logistics

- ▶ Class times
- ▶ Breaks
- ▶ Lunch
- ▶ Restrooms
- ▶ Accessing Wi-Fi
- ▶ Working agreements

Notes:

Course goals

At the end of this course you should be able to:

- ▶ Define Agile software engineering and the underlying values, principles, and practices
- ▶ Apply the test-first principle to create alignment between tests and requirements
- ▶ Create shared understanding with behavior-driven development (BDD)
- ▶ Support effective Program Increment execution
- ▶ Communicate with Agile modeling
- ▶ Design from context for testability
- ▶ Build applications with code and design quality
- ▶ Utilize the test infrastructure for automated testing
- ▶ Collaborate on intentional architecture and emergent design
- ▶ Apply Lean-Agile principles to optimize the flow of value
- ▶ Create an Agile software engineering plan

Notes:

Course map

- ▶ Lesson 1: Introducing the Course
- ▶ Lesson 2: Connecting Principles and Practices to Built-In Quality
- ▶ Lesson 3: Accelerating Flow
- ▶ Lesson 4: Applying Intentional Architecture
- ▶ Lesson 5: Thinking Test-First
- ▶ Lesson 6: Discovering Story Details
- ▶ Lesson 7: Creating a Shared Understanding with Behavior-Driven Development (BDD)
- ▶ Lesson 8: Communicating with Models
- ▶ Lesson 9: Building Systems with Code Quality
- ▶ Lesson 10: Building Systems with Design Quality
- ▶ Lesson 11: Implementing with Quality
- ▶ Lesson 12: Course Review
- ▶ Lesson 13: Becoming a Certified SAFe Professional

SCALED AGILE® © Scaled Agile, Inc.

3

Notes:

The classroom experience

During the three days in class, you will:

- ▶ **Day 1:** Form teams, discuss eXtreme Programming (XP) practices, and create a minimum marketable feature (MMF) for the team's chosen domain
- ▶ **Day 2:** Write and split user stories, create BDD tests, and discuss test doubles, and why communicating with models is important
- ▶ **Day 3:** Discuss code quality and refactor code examples, write lower level tests, and analyze design patterns



SCALED AGILE® © Scaled Agile, Inc.

5

Notes:

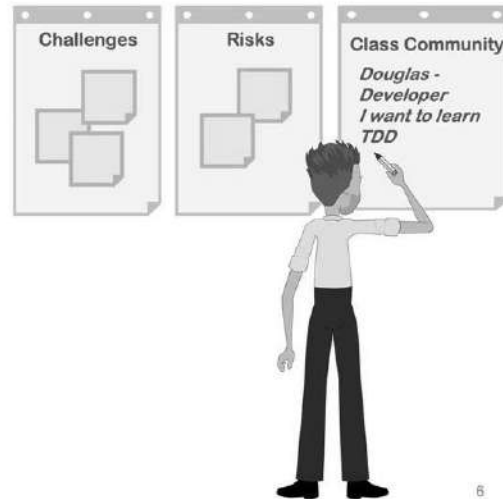


Before we launch: Identify challenges and risks

Prepare



- ▶ **Step 1:** Using yellow sticky notes, identify three challenges and three risks with specifying/defining, developing, testing, and deploying software in your context
- ▶ **Step 2:** Place the sticky notes on the *Challenges* and *Risks* flip chart sheets
- ▶ **Step 3:** Fill in your name, role, organization, Agile and SAFe experience, and why are you taking this course on the *Class Community* flip chart



SCALED AGILE® © Scaled Agile, Inc.

6

Notes:

Lesson 1

Introducing the Course

Learning Objectives:

- 1.1 Define Agile software engineering
- 1.2 Form your Agile software engineering team



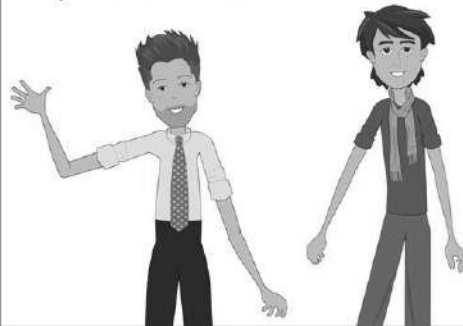
SAFe® Authorized Course Attending this course gives students access to the SAFe® Agile Software Engineer exam and related preparation materials.

Introducing the learning journey

Forming teams around a specific domain (Your own context or *Autonomous Vehicle Parking*) to practice knowledge application and skills



Taking a learning journey with Douglas and Arthur to build your *Agile Software Engineering Improvement Plan*



Notes:



Video: Introducing the course characters

Duration
1 min



<https://vimeo.com/374269424/09c1333a68>

SCALED AGILE® © Scaled Agile, Inc.

7

Notes:



Introductions

Share



- ▶ **Step 1:** Introduce yourself to the person next to you
- ▶ **Step 2:** Share with each other something you know about Agile software engineering

Hi. I'm Arthur.
To me, Agile
software
engineering is ...



SCALED AGILE® © Scaled Agile, Inc.

10

Notes:

1.1 Define Agile software engineering

SCALED AGILE © Scaled Agile, Inc.

12

Notes:

What is software engineering?

The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

— *IEEE Standard Glossary of Software Engineering Terminology*

SCALED AGILE © Scaled Agile, Inc.

Notes:

Agile software engineering

What is Agile software engineering?

- Agile software engineering unifies many well-known principles and practices.

Lean and Agile principles and practices

Behavior-driven development

eXtreme Programming

Code quality

Design patterns and practices

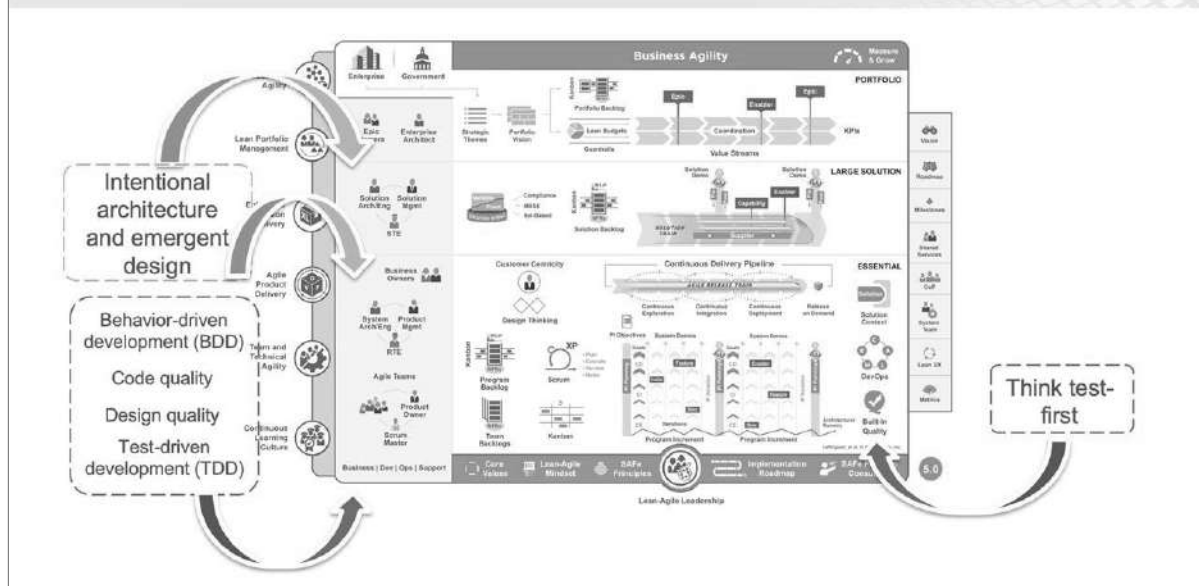
Agile modeling

SCALED AGILE® © Scaled Agile, Inc.

14

Notes:

Agile software engineering in SAFe



Notes:



Notes:



Activity: Create your Agile software engineering team

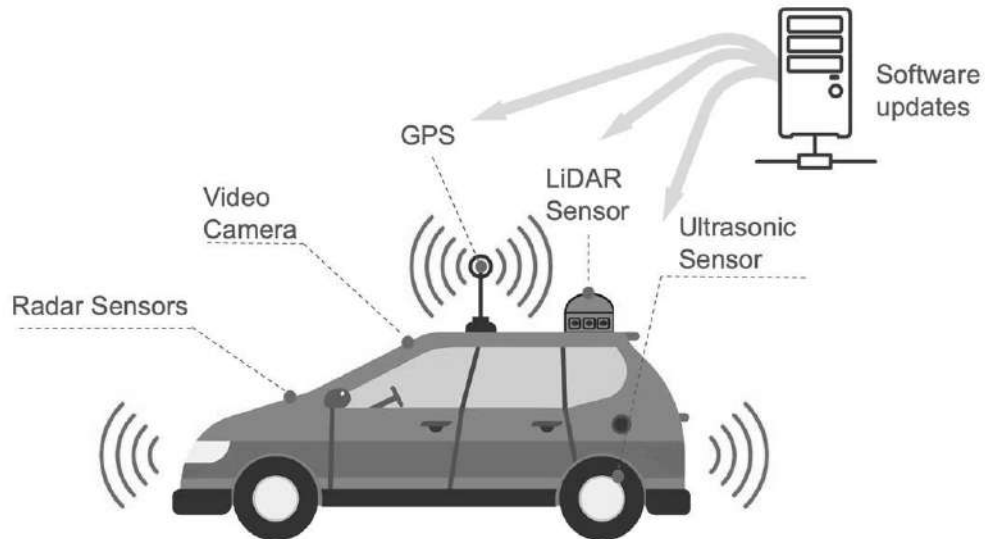


- ▶ **Step 1:** Identify the team you will be working with for the activities throughout the course
 - The team should consist of at least three and not more than five people
- ▶ **Step 2:** Create a team name and a list of your team members and their roles on a flip chart sheet

Notes:

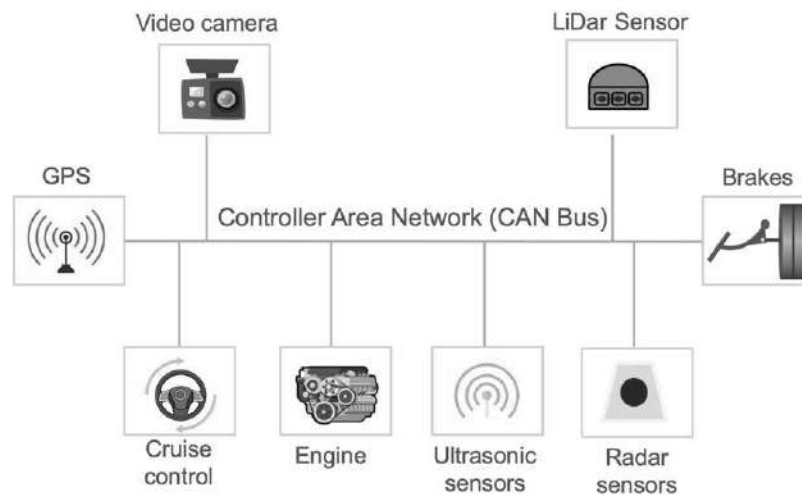
1.2 Form your Agile software engineering team

Class domain: Autonomous vehicle architecture



Notes:

Automobile communication architecture



Notes:

Class domain: Autonomous vehicle requirements (1)

► Speed control



Simple speed control—
set speed and follow it



Determine speed limit
from GPS/Map



Determine speed
limit from signs



Get advance warning of speed limit
when decreases and slow down in time



Alter speed limit based on time
(e.g. school zones)



Notes:

Class domain: Autonomous vehicle requirements (2)

► Speed control with stopping



Identify stop signs from a map or camera
and stop for them


► Speed control with collision control



Drive with speed limit but advise if too
close to vehicle in front



Notes:



Activity: Determine the domain for your team

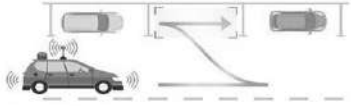
Prepare
2 min

► **Step 1:** Choose the domain for your team


- Option 1: Your own context
- Option 2: Autonomous vehicle parking: Level 2 (aided) and Level 4 (controlled)
- Option 3: Finding parking spaces

► **Step 2:** In your teams, discuss some possible features/stories of your chosen domain (e.g., steering control, parking control, turning radius, etc.)

Parallel parking:
Assisted or controlled



Diagonal parking:
Assisted or controlled



SCALED AGILE® © Scaled Agile, Inc.

22

Notes:



Agile Software Engineering Action Plan



- ▶ **Step 1:** Locate the Agile Software Engineer Action Plan at the end of your workbook
- ▶ **Step 2:** Considering what we have covered, write down at least one improvement item in your Agile Software Engineer Action Plan
- ▶ **Step 3:** Share the item you wrote with the class.



Notes:

Lesson review

In this lesson, you:

- ▶ Defined Agile software engineering and outlined current challenges
- ▶ Formed teams and identified a specific domain for the upcoming activities throughout the course

Notes:

Lesson 2

Connecting Principles and Practices to Built-In Quality

Learning Objectives:

- 2.1 Connect principles to practices
- 2.2 Describe XP practices
- 2.3 Explore key definitions
- 2.4 Describe Built-In Quality
- 2.5 Describe tradeoffs



SAFe® Authorized Course Attending this course gives students access to the SAFe® Agile Software Engineer exam and related preparation materials.



Notes:

2.1 Connect principles to practices

SCALED AGILE® © Scaled Agile, Inc.

4

Notes:

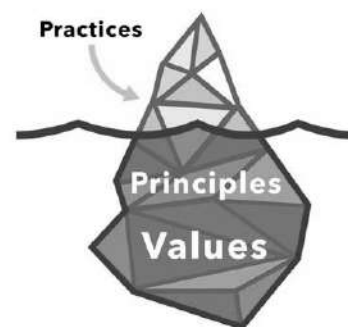
Practices derive from principles and values

Values: What is important, held in high regard

Principles: Fundamental statements based on values

Practices: Action based on principles and values

Enabling Technical Agility for the Lean Enterprise



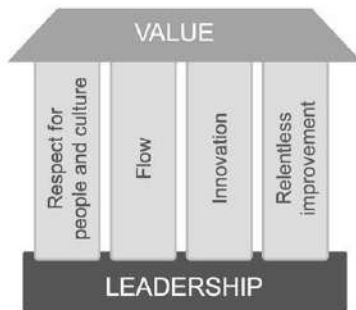
SCALED AGILE® © Scaled Agile, Inc.

5

Notes:

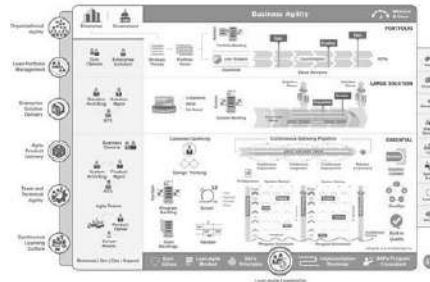
Embracing Lean-Agile mindset

SAFe House of Lean



SCALED AGILE® © Scaled Agile, Inc.

SAFe Core Values



1. Built-In Quality
2. Program execution
3. Alignment
4. Transparency

6

Notes:

Manifesto for Agile software development

- ▶ We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value the Agile Manifesto
- ▶ That is, while there is value in the items on the right, we value the items on the left more.

Agile Manifesto

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

SCALED AGILE® © Scaled Agile, Inc.

7

Notes:

SAFe Principles

- #1 Take an economic view
- #2 Apply systems thinking
- #3 Assume variability; preserve options
- #4 Build incrementally with fast, integrated learning cycles
- #5 Base milestones on objective evaluation of working systems
- #6 Visualize and limit WIP, reduce batch sizes, and manage queue lengths
- #7 Apply cadence, synchronize with cross-domain planning
- #8 Unlock the intrinsic motivation of knowledge workers
- #9 Decentralize decision-making
- #10 Organize around value


SCALED AGILE® © Scaled Agile, Inc.

8

Notes:



Notes:



Discussion: eXtreme Programming (XP) practices


Discuss 2 min

Share 2 min

► **Step 1:** At your table, discuss eXtreme Programming (XP) development practices. Consider the following questions:

- Have you heard of XP?
- What XP development practices can you recall?

► **Step 2:** On a flip chart sheet, list as many XP development practices as you can.



XP Practices

1. _____

2. _____

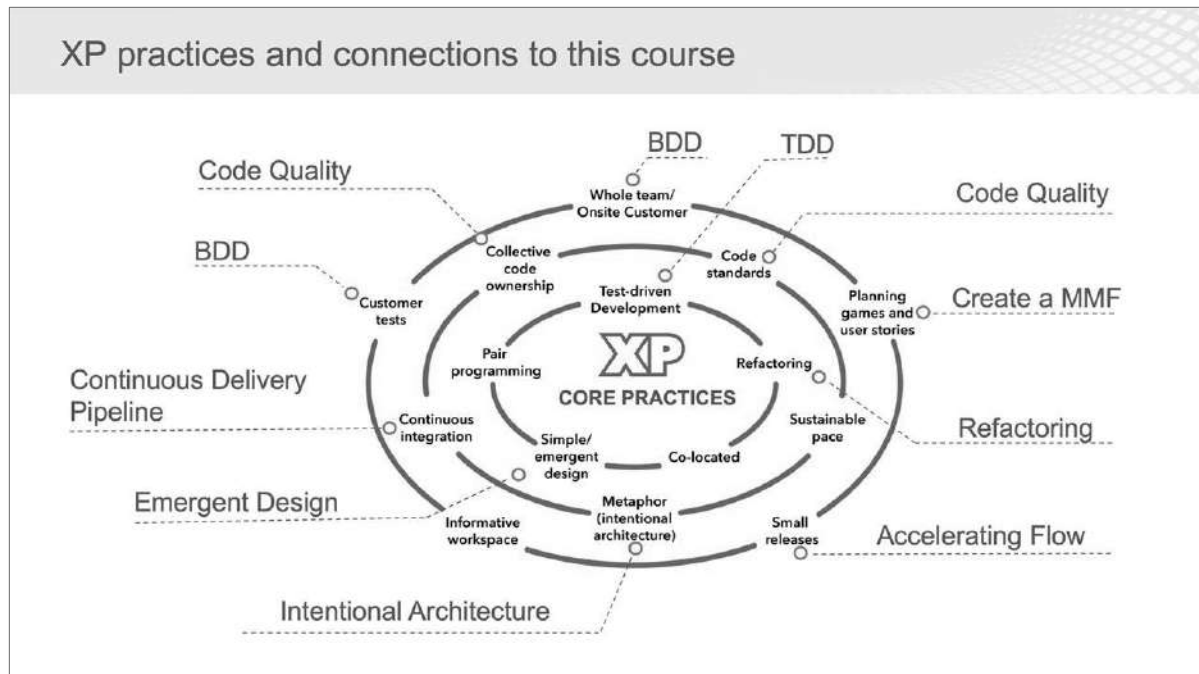
3. _____

SCALED AGILE® © Scaled Agile, Inc.

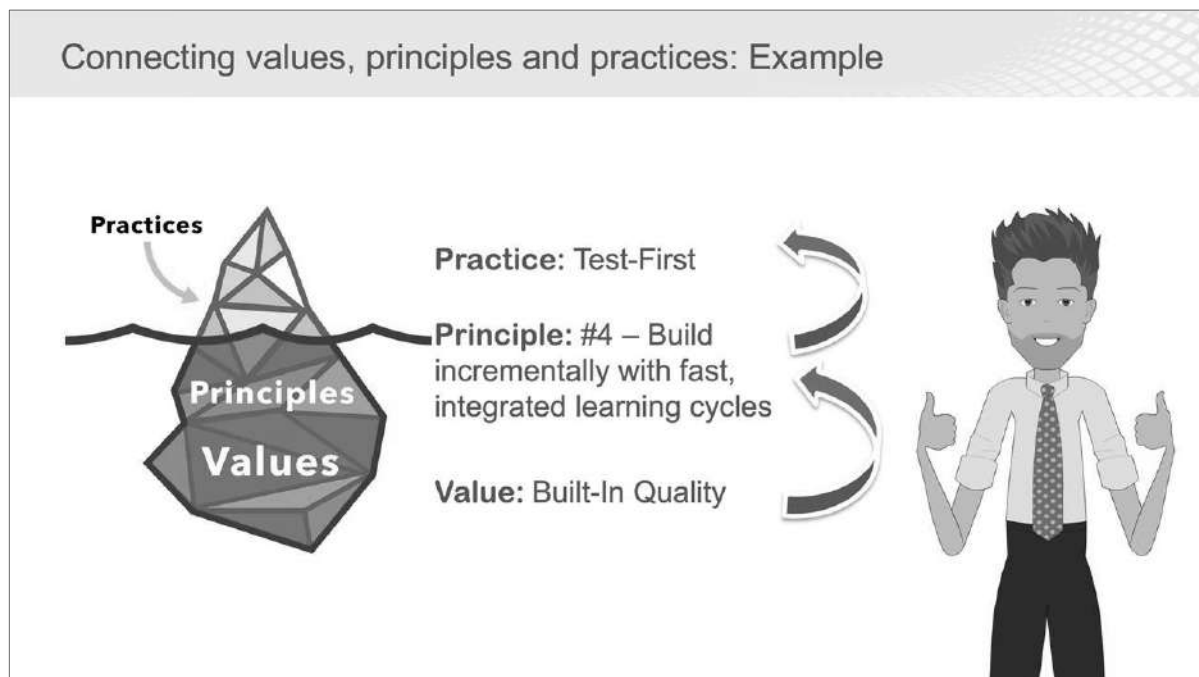
10

Notes:


2.2 Describe XP practices



Notes:



Notes:



Activity: Connect XP core practices to SAFe Principles

Prepare
5 min

Share
5 min

- ▶ **Step 1:** Working at your table, discuss which of SAFe's principles have a strong connection to the XP core practices
- ▶ **Step 2:** In your workbook, using the XP Practices and SAFe Principles Matrix, draw a line to show the connections

SCALED AGILE® © Scaled Agile, Inc.

13

Notes:

SAFe Principles

#1 Take an economic view

#2 Apply systems thinking

#3 Assume variability; preserve options

#4 Build incrementally with fast, integrated learning cycles

#5 Base milestones on objective evaluation of working systems

#6 Visualize and limit WIP, reduce batch sizes, and manage queue lengths

#7 Apply cadence, synchronize with cross-domain planning

#8 Unlock the intrinsic motivation of knowledge workers

#9 Decentralize decision-making

#10 Organize around value

© Scaled Agile, Inc.

XP Core Practices

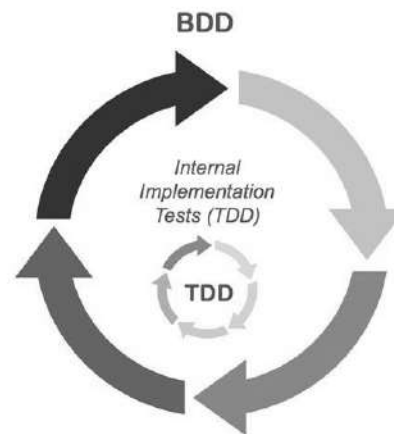
Small releases	#6
Refactoring	
Customer tests	
Code standards	
Co-located teams	
Emergent Design	
Sustainable pace	
Pair Programming	
Intentional Architecture	
Continuous integration	
Test-Driven Development	
Collective code ownership	
Informative workspace	
Whole Team/Onsite Customer	
Planning games and User Stories	



Notes:

Key definitions throughout the course

- ▶ Benefit hypothesis validation
- ▶ Behavior-driven development (BDD)
- ▶ Test-driven development (TDD)



SCALED AGILE® © Scaled Agile, Inc.

15

Notes:

Benefit Hypothesis Validation -

Revolves around checking that users are actually utilizing a feature.

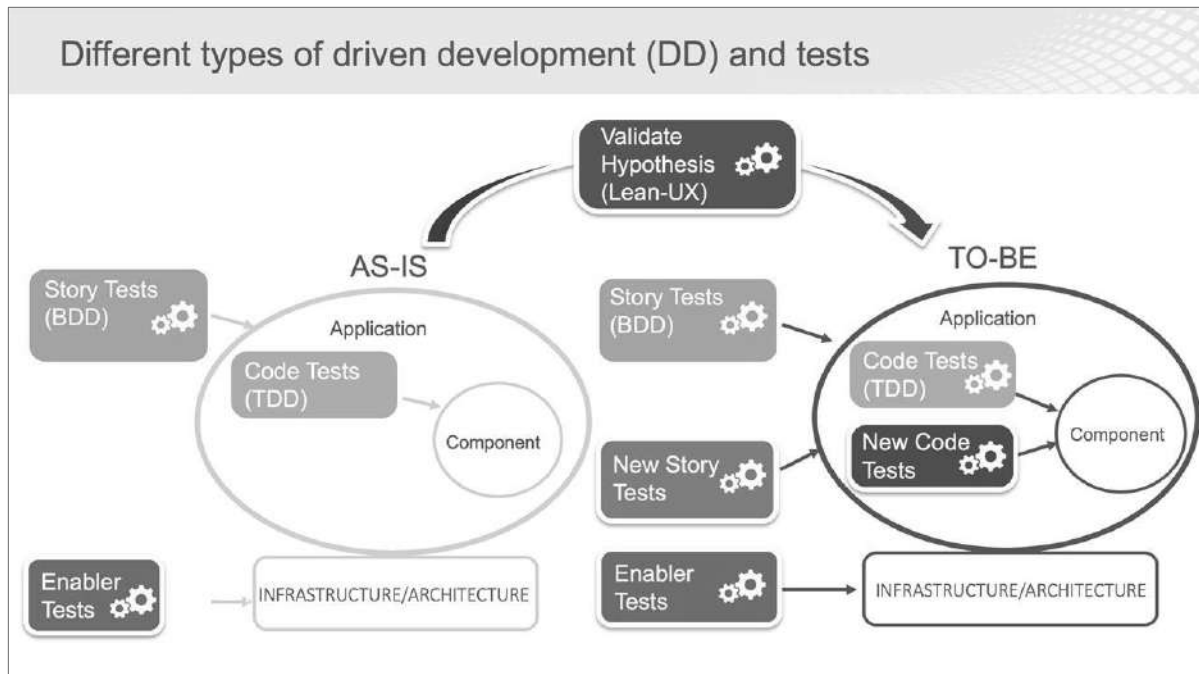
Behavior-Driven Development (BDD) -

Focuses on understanding external requirements by creating scenarios of operation which then get formulated into tests.

Test-Driven Development (TDD) -

Developers write an automated unit test first which specifies an internal behavior, run the test to observe the failure, and then write the minimum code necessary to pass the test


2.3 Explore key definitions



Notes:



Notes:



Discussion: What is quality?

Discuss
2 min

Share
2 min


► **Step 1:** On a sticky note, write your definition of quality.

► **Step 2:** Pair with another person. Discuss your definitions and try to come up with one that generally agrees with both your definitions.

► **Step 3:** Join your pair with another to form a small group. As a group, discuss the definitions you formed in Step 2 and try to come up with one common definition.

► **Step 4:** As a group, share your definition with the class.


Quality



SCALED AGILE® © Scaled Agile, Inc.

18

Notes:




Activity: What are the attributes of quality?

Prepare
3 min

Share
2 min

- ▶ **Step 1:** Write down on individual sticky notes an attribute of quality as you see it.
- ▶ **Step 2:** Place the sticky notes with attributes on the board. Try to place the matching ones together.
- ▶ **Step 3:** Read out the items from the most matches to the least matches.



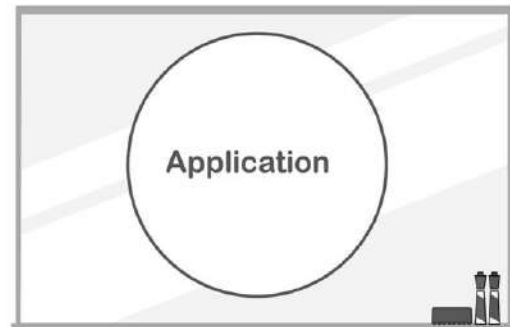
SCALED AGILE® © Scaled Agile, Inc.

19

Notes:

Application qualities

- ▶ External (execution) application qualities
 - Meets user's needs
 - Performance
 - Security
 - Scalability
 - Usability
 - Reliability
 - Efficiency
 - Testability



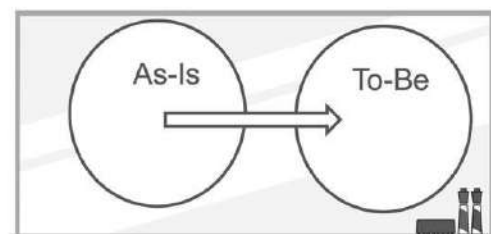
SCALED AGILE[®] © Scaled Agile, Inc.

21

Notes:

Application qualities

- ▶ Internal (evolution) application qualities
 - Maintainability/modifiability
 - Ease of going from *As-Is* to *To-Be*
 - Testability of internal components



SCALED AGILE[®] © Scaled Agile, Inc.

22

Notes:



Notes:



Discussion: What tradeoffs have you made?



- ▶ **Step 1:** At your table, discuss:
 - Is there ever a conflict between all the values, principles, practices, and quality?
 - What tradeoffs have you made in the past?
 - Have you ever performed a cost-benefit analysis on aspects of quality?
- ▶ **Step 2:** Share with the class.

Notes:

Tradeoffs summary

There are no always true engineering decisions

- ▶ Everything exists in a context
- ▶ Every context contains different tradeoffs

Notes:



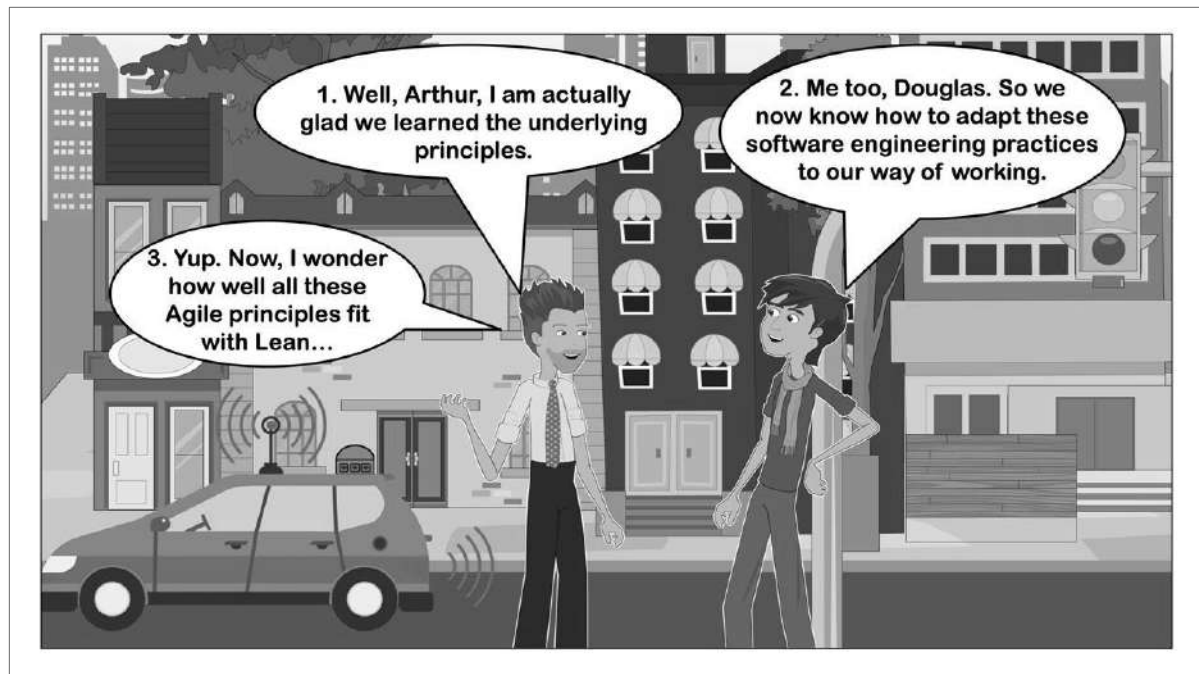
Agile Software Engineering Action Plan



- **Step 1:** Locate the Agile Software Engineer Action Plan at the end of your workbook
- **Step 2:** Considering what we have covered, write down at least one improvement item in your Agile Software Engineer Action Plan
- **Step 3:** Share the item you wrote with the class.



Notes:



Notes:

Lesson review

In this lesson, you:

- ▶ Connected principles to practices
- ▶ Described XP practices
- ▶ Explored key definitions, Built-in Quality, and tradeoffs

Notes:

Lesson 3

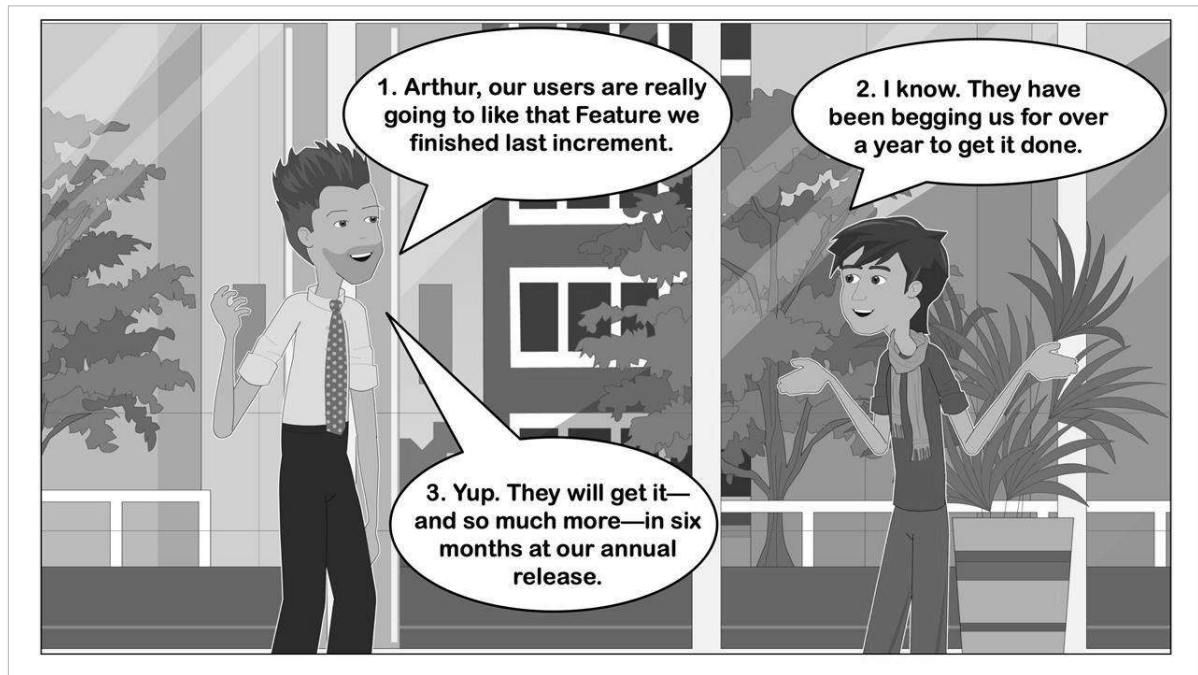
Accelerating Flow

Learning Objectives:

- 3.1 Outline the flow of value
- 3.2 Validate the benefit hypothesis



SAFe® Authorized Course Attending this course gives students access to the SAFe® Agile Software Engineer exam and related preparation materials.



Notes:

3.1 Outline the flow of value

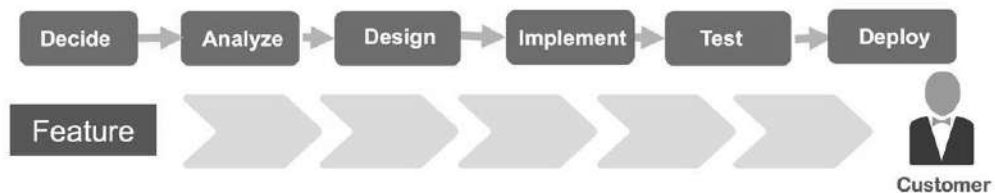
SCALED AGILE © Scaled Agile, Inc.

4

Notes:

Value Streams accelerate delivery of value to Customers

- ▶ Value Streams optimize the sequence of steps necessary to release Features to Customers
- ▶ Value Streams include all resources:
 - The people who do the work
 - The systems
 - The flow of information and materials




SCALED AGILE © Scaled Agile, Inc.

5

Notes:

3.1 Outline the flow of value



Activity: Diagram current flow for a feature

Prepare
12 min

Share
3 min

- ▶ **Step 1:** At your table, choose a context from your own domain of work (one per table).
- ▶ **Step 2:** Using a flip chart sheet, diagram the high-level steps (between 8 to 15) necessary to release a Feature to the Customer. Consider the work needed to:
 - Analyze, design, build, test, validate, etc. the Feature
 - Make the Feature available to the Customer
- ▶ **Step 3:** Share with the class.

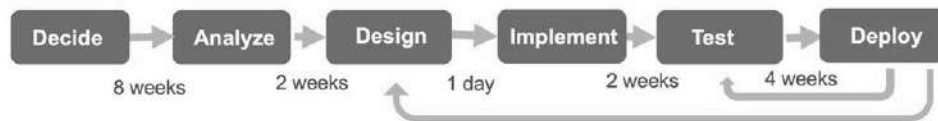
Note: Save the flip chart sheet. It will be used later in the course.

SCALED AGILE® © Scaled Agile, Inc. 6

Notes:

Identifying waste: Delays and loopbacks


- ▶ Not all steps in the flow occur sequentially or immediately after the completion of the previous step
- ▶ Delays and loopbacks slow value delivery and are forms of waste in Lean



SCALED AGILE® © Scaled Agile, Inc.

7

Notes:




Activity: Identify delays and loopbacks

Prepare
5 min

Share
5 min

- ▶ **Step 1:** Go back to the chart you created for the flow of a feature in the previous activity.
- ▶ **Step 2:** Identify and examine some of the delays and loopbacks in the flow:
 - Indicate the average delay time between steps and some of the common loopbacks.
 - Based on the delays and loopbacks, how quickly can your organization deliver Features?
- ▶ **Step 3:** Explain the delays and loopbacks to the class.



SCALED AGILE® © Scaled Agile, Inc.

8

Notes:


3.2 Validate the benefit hypothesis

SCALED AGILE © Scaled Agile, Inc.

9


Notes:

3.2 Validate the benefit hypothesis



Activity: What is business value?

Share
5 min



How is value defined by your business?

Business value is...

SCALED AGILE® © Scaled Agile, Inc.

10

Notes:

3.2 Validate the benefit hypothesis

Examples of business value

- ▶ Decreased risk impact
- ▶ Decreased expenses
- ▶ Customer promoters, satisfiers, detractors
- ▶ Brand recognition
- ▶ Increased revenue
- ▶ Increased market share
- ▶ Increased quality
- ▶ Higher productivity
- ▶ Enter new market

SCALED AGILE® © Scaled Agile, Inc.

11

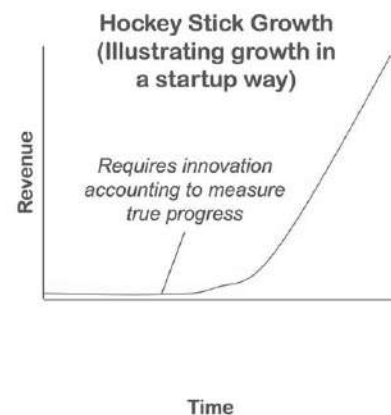
Notes:

Assessing value for innovation requires new measures

Innovation Accounting is a way of evaluating progress when all the metrics typically used in an established company (revenue, customer, ROI, market share) are effectively zero.

—The Startup Way

- ▶ In innovative spaces, traditional measures are often lagging indicators
- ▶ Measure progress as knowledge gained directly from Customer



SCALED AGILE® © Scaled Agile, Inc.

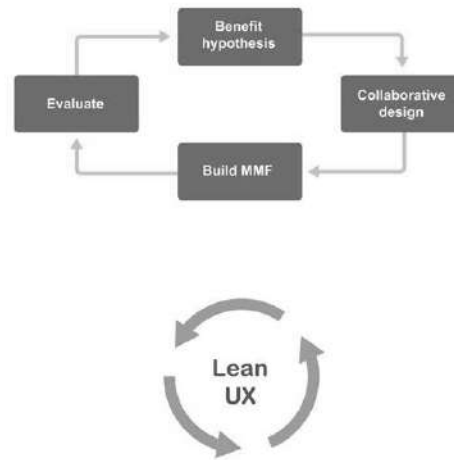
12

Notes:

3.2 Validate the benefit hypothesis

Using benefit hypothesis to validate business value

- ▶ Every requirement is an assumption about business value
- ▶ Use experimentation to test assumptions on Customers and their behavior
- ▶ Learn what users truly desire based on their behavior
- ▶ This is part of Lean UX



SCALED AGILE® © Scaled Agile, Inc.

13

Notes:

Testing hypothesis with a minimum marketable feature (MMF)

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

—Agile Manifesto

Minimum Marketable Feature (MMF):

DEFINITION 

Smallest piece of functionality with intrinsic business/market value



PURPOSE

Release functionality sooner, learn from users faster

SCALED AGILE® © Scaled Agile, Inc.

14

Notes:

3.2 Validate the benefit hypothesis

A benefit hypothesis template for business value

- ▶ MMFs need both a benefit hypothesis and metrics.
- ▶ Template:
 - ▶ We think <Capability> will produce <outcome> as measured by <Metric>

Examples:

- We think <Cruise Control> will sell <more cars> as measured by <at least 50% of car sales will include Cruise Control> (longer term)
- We think that Cruise Control with Speed Limit Finder will be used by drivers 90% of the time as measured by telemetry (shorter term)

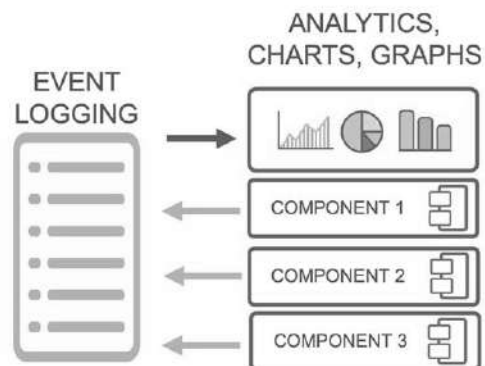
SCALED AGILE® © Scaled Agile, Inc.

15

Notes:

Use measures and telemetry to evaluate hypothesis

- ▶ Example of measures:
 - Features used
 - Time spent on features
 - Paths through applications
- ▶ Telemetry captures events to analyze these measures.



SCALED AGILE® © Scaled Agile, Inc.

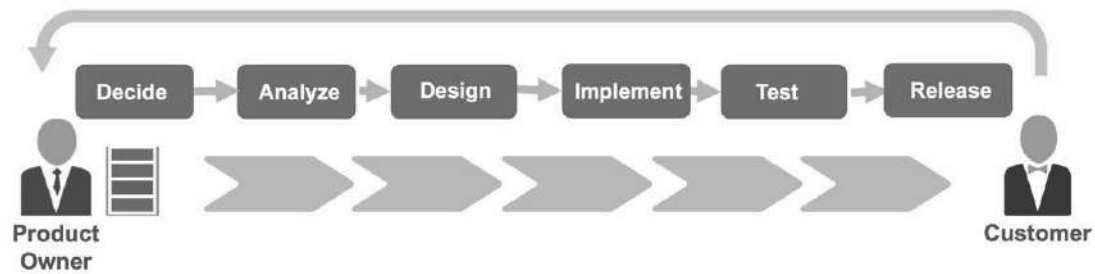
16

Notes:

3.2 Validate the benefit hypothesis

Observe changes in Customer behavior

- ▶ Measure and analyze results of hypothesis to draw conclusions
- ▶ Final feedback loop of full Value Stream

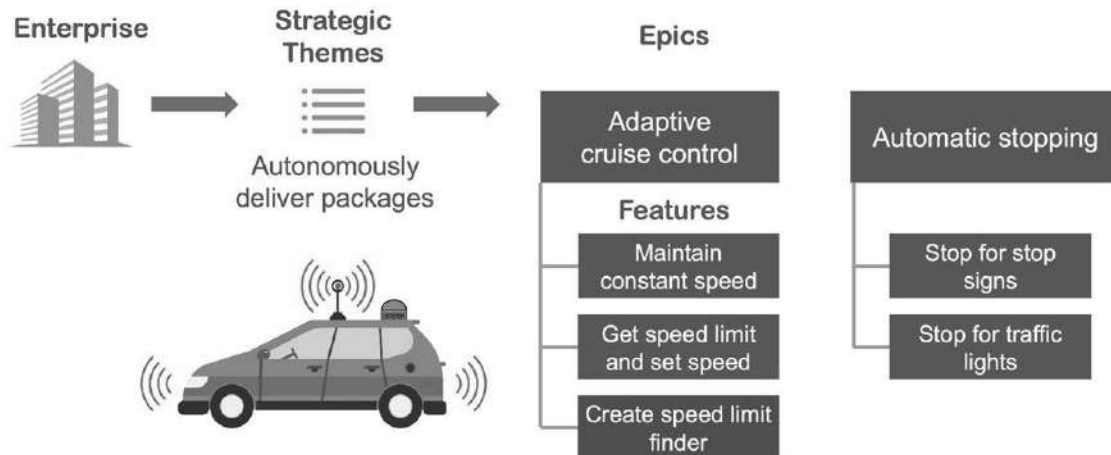


SCALED AGILE® © Scaled Agile, Inc.

17

Notes:

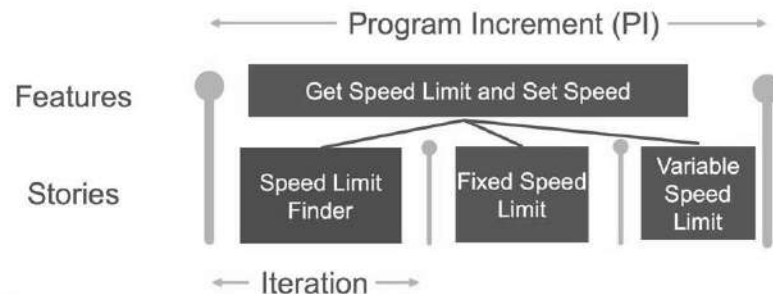
Where do MMFs come from?



Notes:

Features and Stories incrementally deliver value

- ▶ MMF is smallest releasable Feature
- ▶ User stories implement a small, vertical slice of a Feature's functionality
- ▶ Features may exist without a parent Epic and Stories may exist without a parent Feature




SCALED AGILE® © Scaled Agile, Inc.

19

Notes:

3.2 Validate the benefit hypothesis



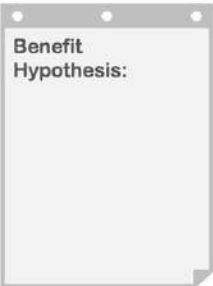
Activity: Minimum marketable feature (MMF)

Prepare
10 min

Share
5 min

Using a flip chart sheet, create an MMF with your team (choose your own domain or the *Autonomous Vehicle Parking*):

- **Step 1:** Create a benefit hypothesis for the MMF:
 - How can you prove or disprove the hypothesis?
 - What do you want to learn about how users are using the Features?
 - What information would you need to record in order to acquire that learning?
- **Step 2:** Be prepared to share with the class.



Benefit Hypothesis:

SCALED AGILE © Scaled Agile, Inc.

20

Notes:



Agile Software Engineering Action Plan



- **Step 1:** Locate the Agile Software Engineer Action Plan at the end of your workbook
- **Step 2:** Considering what we have covered, write down at least one improvement item in your Agile Software Engineer Action Plan
- **Step 3:** Share the item you wrote with the class.



Notes:

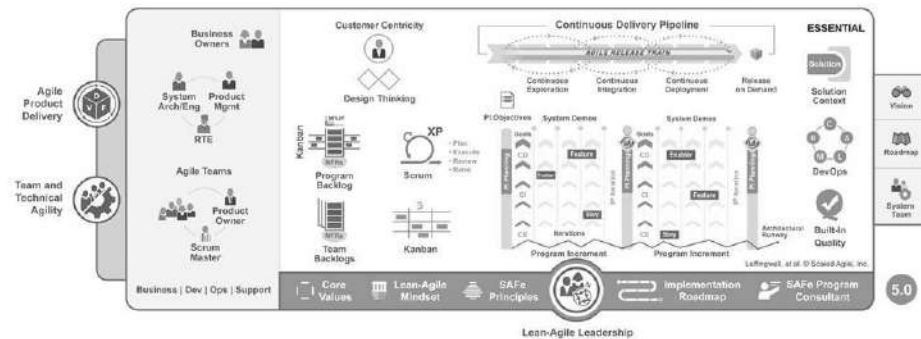


Discussion: Accelerating flow in SAFe



► Discuss:

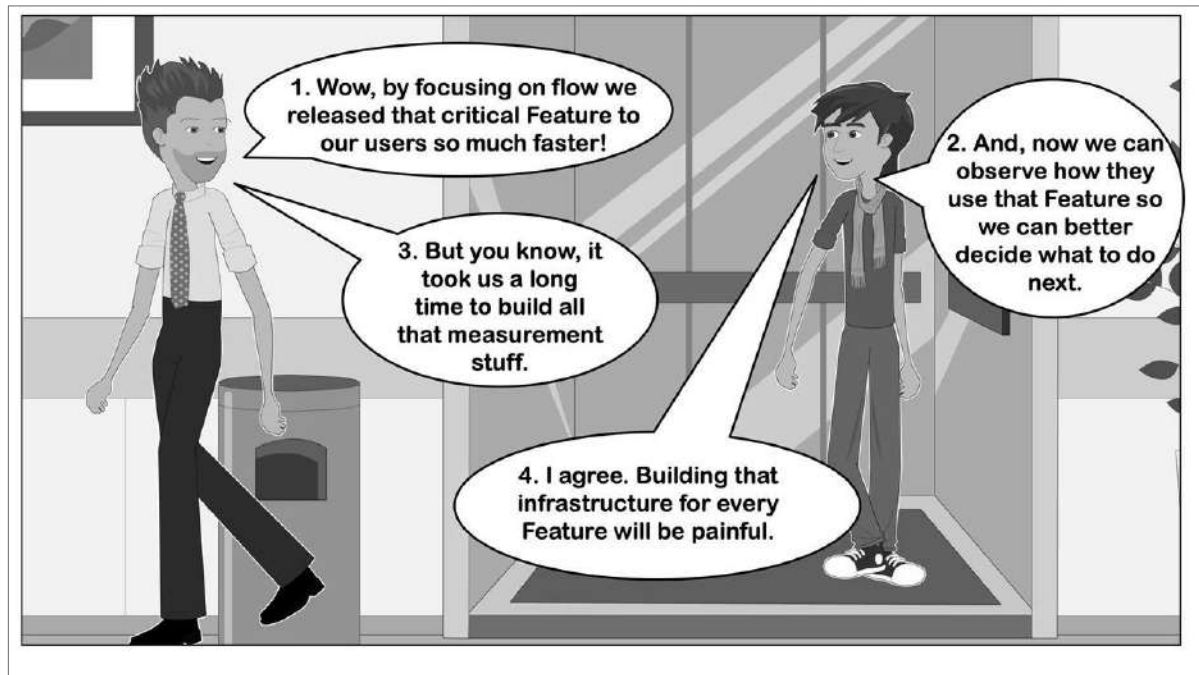
- Where in SAFe do we discuss accelerating flow?
- Who in SAFe would be responsible for accelerating flow?



SCALED AGILE® © Scaled Agile, Inc.

22

Notes:



Notes:

Lesson review

In this lesson, you:

- ▶ Outlined the flow of value
- ▶ Validated the benefit hypothesis

Notes:

Lesson 4

Applying Intentional Architecture

Learning Objectives:

- 4.1 Apply systems thinking
- 4.2 Explain the role of Agile architecture in supporting Lean-Agile development
- 4.3 Architect and design for testability



SAFe® Authorized Course Attending this course gives students access to the SAFe® Agile Software Engineer exam and related preparation materials.

4.1 Apply systems thinking

SCALED AGILE © Scaled Agile, Inc.

4

Notes:

Systems thinking

Systems thinking is a discipline for seeing wholes. It is a framework for seeing interrelationships rather than things, for seeing patterns of change rather than static 'snapshots'.

— Peter Senge

- ▶ Take a systems view (big picture)
 - Decisions should be congruent with the big picture
- ▶ Don't sub-optimize, optimize the whole

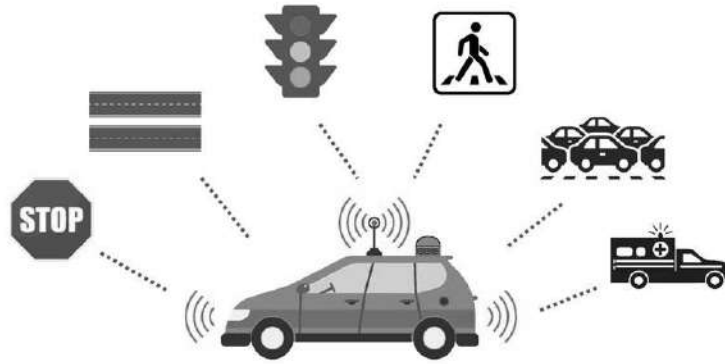
SCALED AGILE © Scaled Agile, Inc.

5

Notes:

Consider the Solution's operational context

- ▶ Stop signs
- ▶ Street lines
- ▶ Traffic lights
- ▶ Pedestrians
- ▶ Other traffic
- ▶ Emergency vehicles



SCALED AGILE[®] © Scaled Agile, Inc.

6

Notes:

Considers the Solution's development context


- ▶ Other development teams
- ▶ Operations
- ▶ Business and marketing determine Features
- ▶ Suppliers
- ▶ Regulatory compliance
- ▶ Manufacturability



SCALED AGILE[®] © Scaled Agile, Inc.

7

Notes:



Activity: Apply Systems Thinking to context

Prepare 5 min


Share 3 min

► **Step 1:** In your teams, considering the domain (own or *Autonomous Vehicle Parking*), think about what elements are in the:


- Operational context
- Development context

► **Step 2:** On a flip chart sheet or a whiteboard, record a list of elements for each context.

Operational context



Development context



SCALED AGILE® © Scaled Agile, Inc.

8

Notes:


4.2 Explain the role of Agile architecture in supporting Lean-Agile development

SCALED AGILE © Scaled Agile, Inc.

9

Notes:

4.2 Explain the role of Agile architecture in supporting Lean-Agile development



Discussion: Making architecture decisions

Discuss
5 min

Share
5 min

- **Step 1:** At your table discuss the difference between architecture and design. Write the definition of each on a flip chart sheet.
- **Step 2:** Consider who makes the decisions on architecture and how are they communicated.
- **Step 3:** Look at another team's definitions. Do you agree with it?



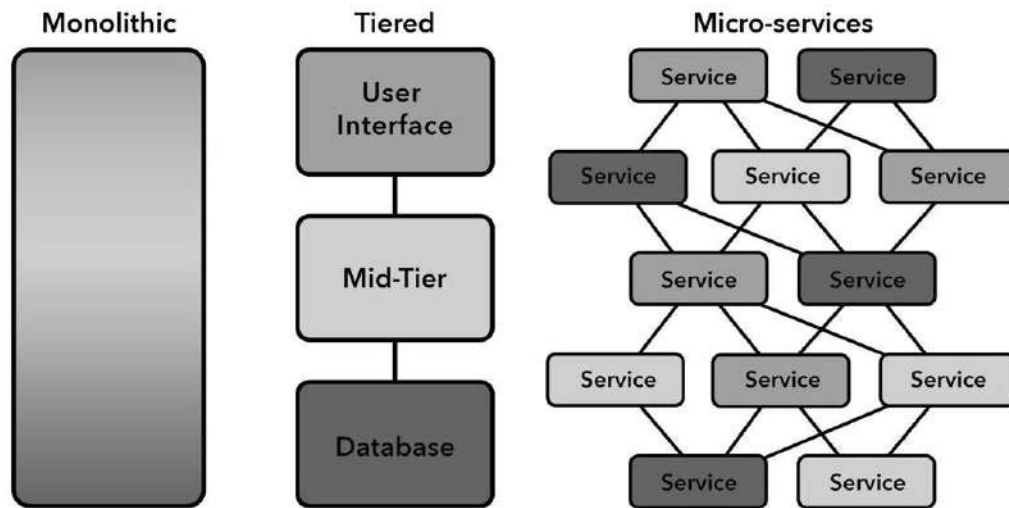
Architecture or design??

SCALED AGILE © Scaled Agile, Inc.

10


Notes:

Some architectural patterns



Notes:

4.2 Explain the role of Agile architecture in supporting Lean-Agile development

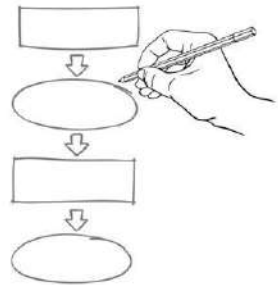


Activity: Sketching architecture

Sketch
5 min

Discuss
5 min

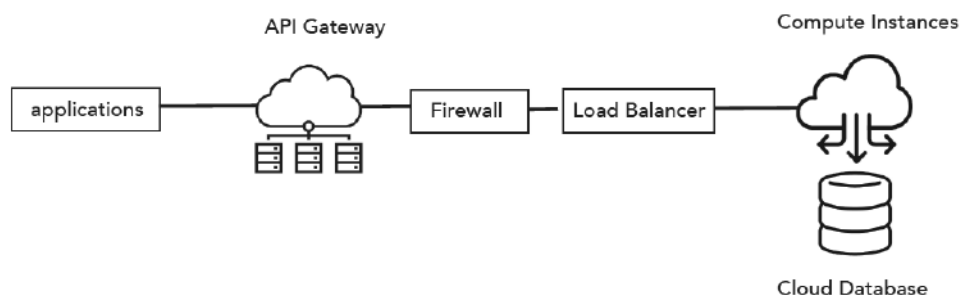
- **Step 1:** As a team, sketch the architecture for a system you know well. You can use the same context that you used to diagram the flow for a Feature.
- **Step 2:** Consider the following:
 - Does the architecture support the release of individual features? Why, or why not?
- **Step 3:** Share with the class.



SCALED AGILE® © Scaled Agile, Inc.

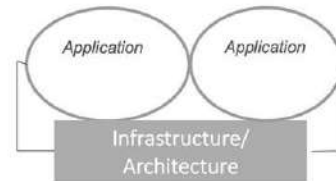
12

Notes:



Emergent design and intentional architecture

- ▶ Plan globally, develop locally
 - Enable incremental value delivery by balancing emergent design and intentional architecture
- ▶ **Emergent design** – teams grow the system design as user stories require
- ▶ **Intentional architecture** – fosters team alignment and defines Architectural Runway



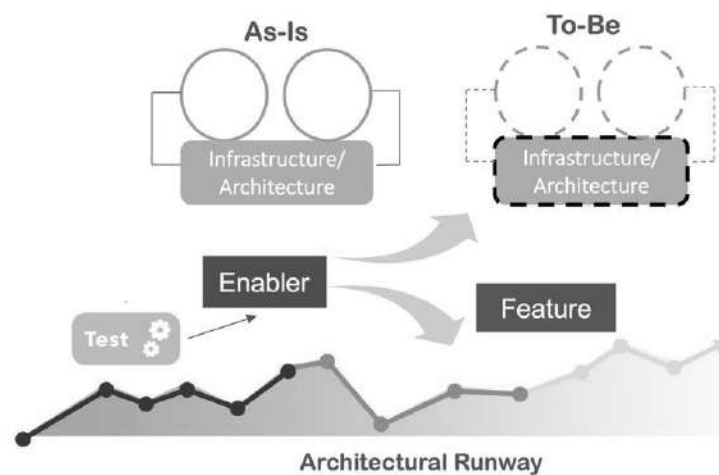
SCALED AGILE® © Scaled Agile, Inc.

13

Notes:

Architectural Runway enables near-term features

- ▶ Enablers build up the runway by evolving the architecture
- ▶ Features use the runway to deliver faster
- ▶ Enablers also have tests, which often support non-functional requirements



SCALED AGILE® © Scaled Agile, Inc.

14

Notes:

Applying intentional and collaborative architecture

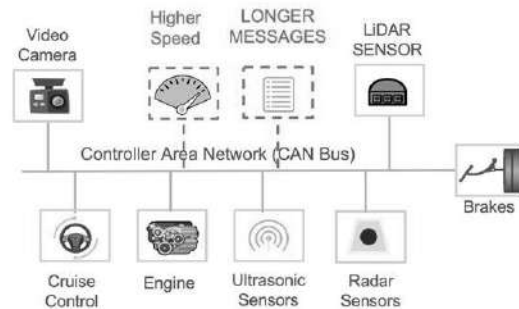
- Consistency is simplicity—can easily understand and maintain a system

A foolish consistency is the hobgoblin of little minds.

— Emerson

- Must simultaneously evolve architecture while continuing to support current user needs

Two components require capabilities beyond the current architecture. What are strategies to address this problem?



Notes:

4.3 Architect and design for testability

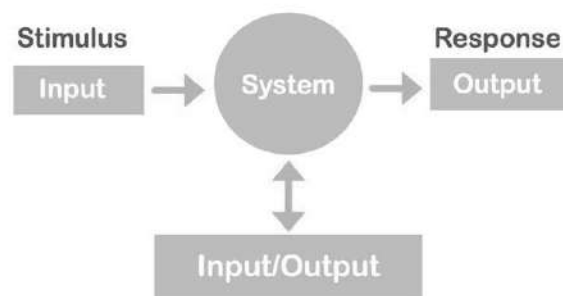
SCALED AGILE © Scaled Agile, Inc.

16

Notes:

Template for context diagram

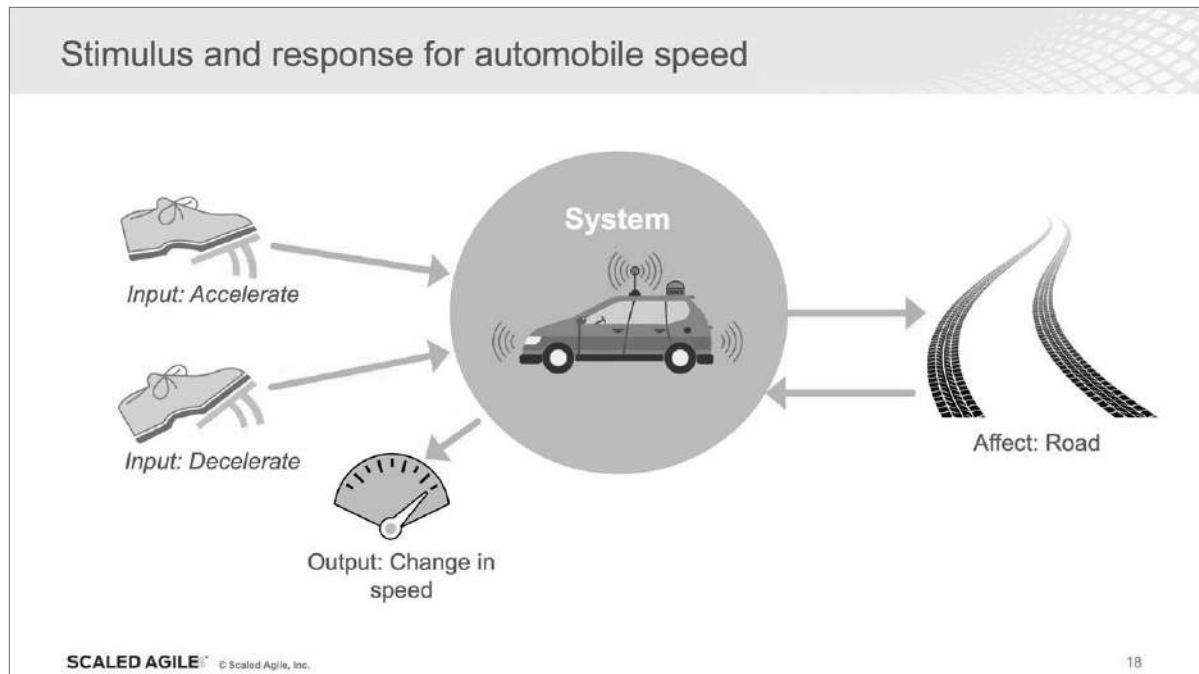
- ▶ External stimulus into system causes some response:
 - Stimulus: From users, services, other systems
 - Response: Change of state, return of information, effects on environment




SCALED AGILE © Scaled Agile, Inc.

17

Notes:



Notes:

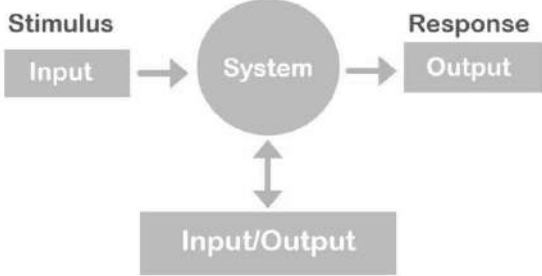


Activity: Draw a context diagram

Create 5 min

Share 5 min

- ▶ **Step 1:** As a team, create a context diagram for the domain (your own or the *Autonomous Vehicle Parking*).
- ▶ **Step 2:** Consider the interconnections between that system and the external world (other applications, end users, services, and more).
 - What inputs does the system receive?
 - What outputs do those inputs cause?
- ▶ **Step 3:** Share with the class.



```
graph LR; Stimulus[Stimulus Input] --> System((System)); System --> Response[Response Output]; System <--> IO[Input/Output];
```

SCALED AGILE® © Scaled Agile, Inc.

19

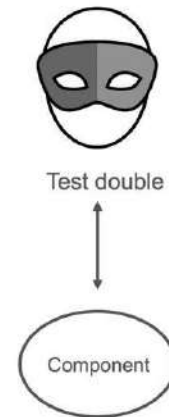
Notes:

What are Test Doubles?

Definition: A simplified version of a depended-upon component that is slow, expensive, or random.

* Stands in like a stunt double for an actor/actress. Test doubles can be also Mocks, Stubs, or Fakes.

- ▶ Facilitate testing the system under test (SUT)
- ▶ Provide stimulus and/or response to SUT
 - Always use the same stimulus/response or
 - Use test to configure stimulus/response

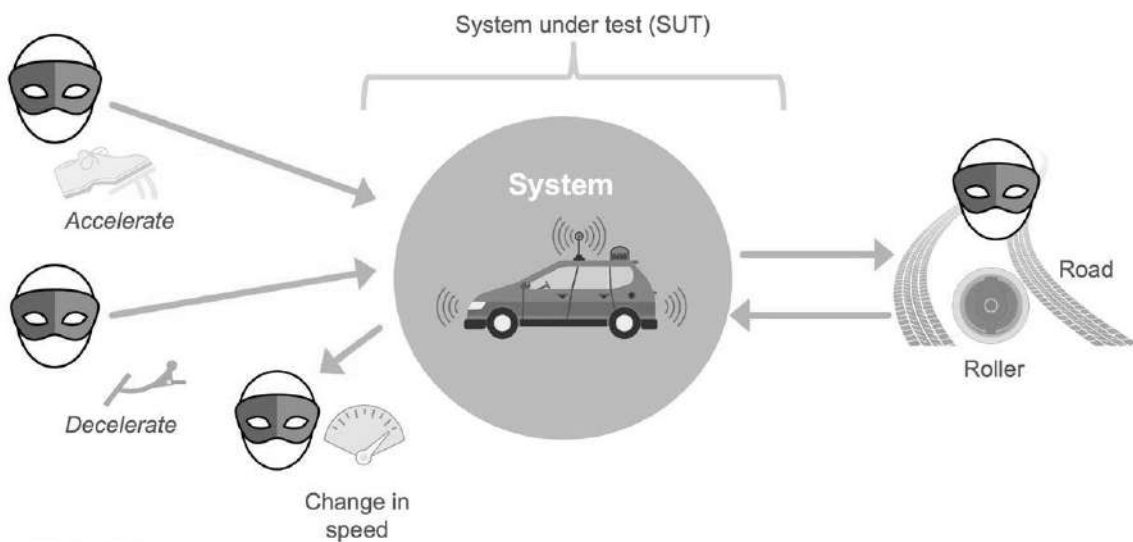


SCALED AGILE® © Scaled Agile, Inc.

20

Notes:

Test doubles for automobile speed

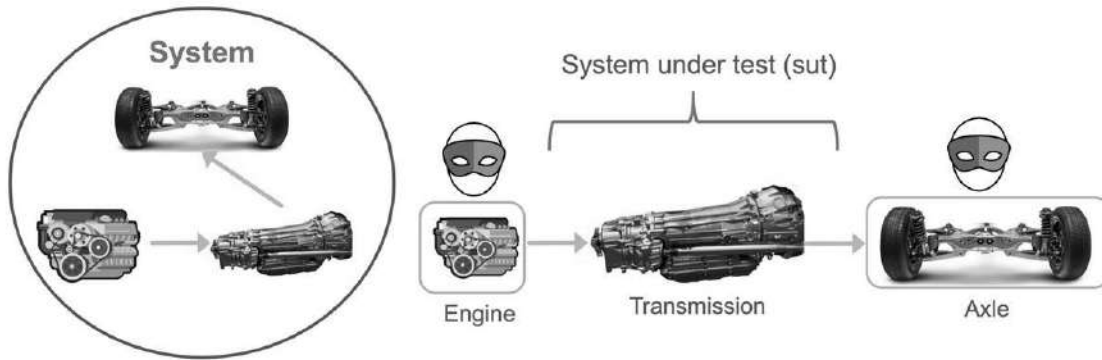


SCALED AGILE® © Scaled Agile, Inc.

21

Notes:

Test doubles for components



Notes:



Discussion: Applying test doubles

Discuss



- ▶ **Step 1:** Refer to the context diagram you previously created for the system (your domain or the autonomous vehicle parking).
- ▶ **Step 2:** Considering the system's testability, discuss the following questions:
 - Do you currently apply test doubles for anything that the system operates with on the outside?
 - If not, how might you consider applying them?



Notes:



Agile Software Engineering Action Plan



- **Step 1:** Locate the Agile Software Engineer Action Plan at the end of your workbook
- **Step 2:** Considering what we have covered, write down at least one improvement item in your Agile Software Engineer Action Plan
- **Step 3:** Share the item you wrote with the class.



Notes:



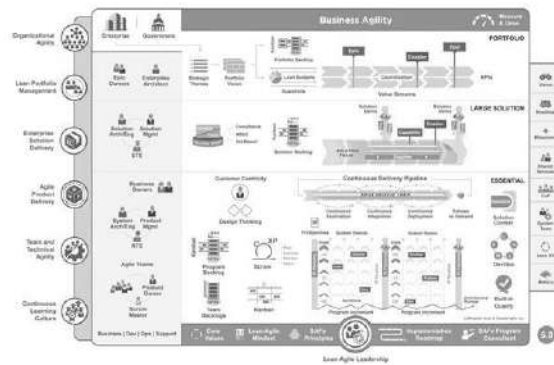
Discussion: Connection to SAFe

Discuss

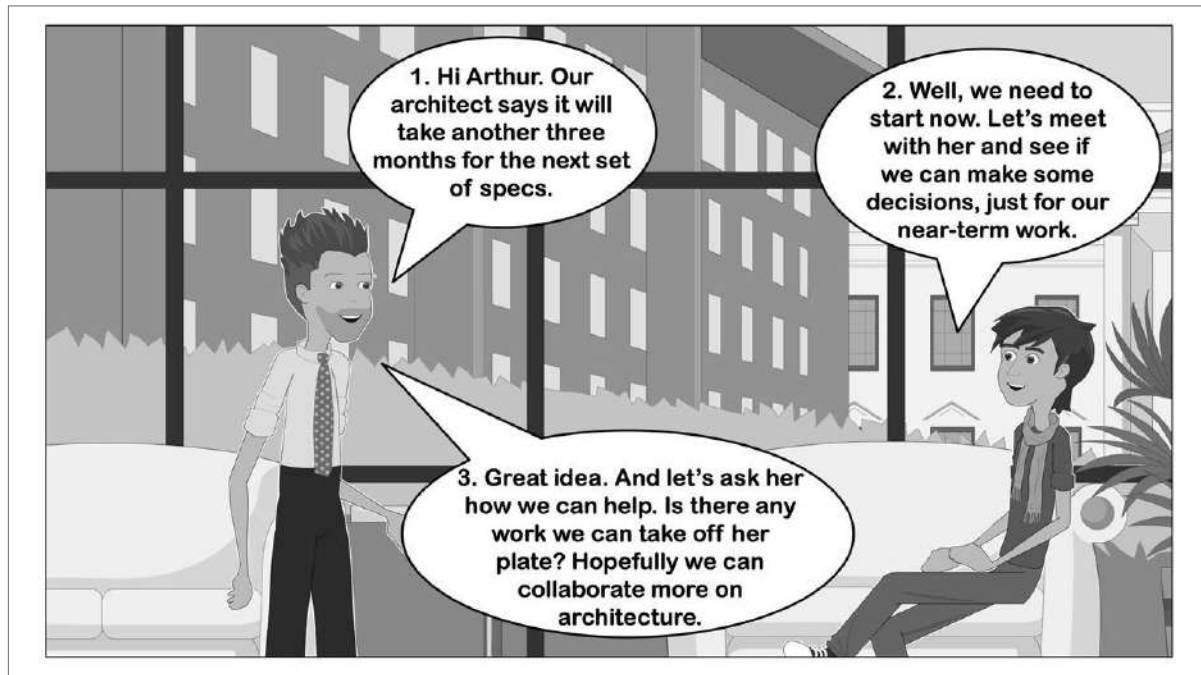


► Discuss:

- What roles in SAFe collaborate on architecture and design?



Notes:



Notes:

Lesson review

In this lesson, you:

- ▶ Applied systems thinking
- ▶ Explained the role of Agile architecture in supporting Lean-Agile development
- ▶ Architected and designed for testability

Notes:

Lesson 5

Thinking Test-First

Learning Objectives:

- 5.1 Shift testing left
- 5.2 Explain the Agile testing matrix
- 5.3 Describe the role of nonfunctional requirements (NFRs)
- 5.4 Build quality in throughout the pipeline



SAFe® Authorized Course Attending this course gives students access to the SAFe® Agile Software Engineer exam and related preparation materials.



Notes:

Video: Why Test-First?

Duration
1 min

SAFe insights

<https://vimeo.com/374271191/f87b3dbe0c>

SCALED AGILE® © Scaled Agile, Inc.

94

Notes:

5.1 Shift testing left

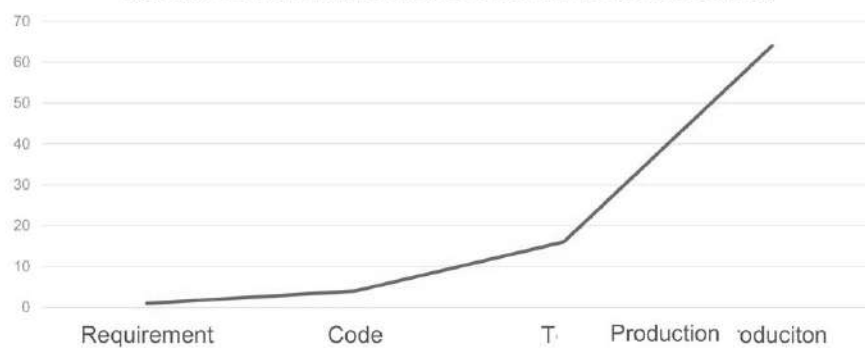
SCALED AGILE © Scaled Agile, Inc.

5

Notes:

Why invest in finding defects earlier?

Relative cost of a defect based on when it was discovered

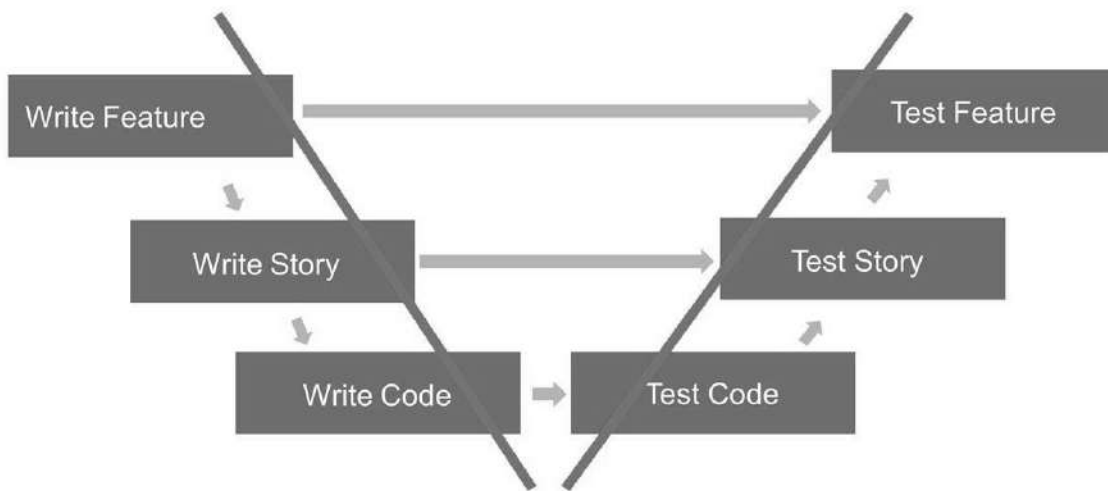


This is relative cost. Do you know what your relative costs are?

Notes:

5.1 Shift testing left

Traditional testing (V-Model) delays feedback

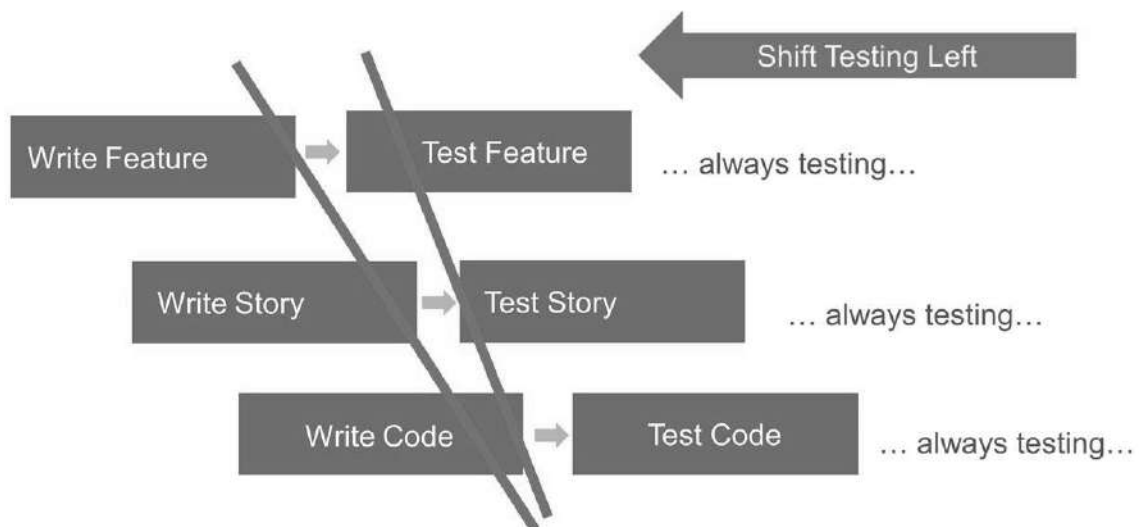


SCALED AGILE® © Scaled Agile, Inc.

7

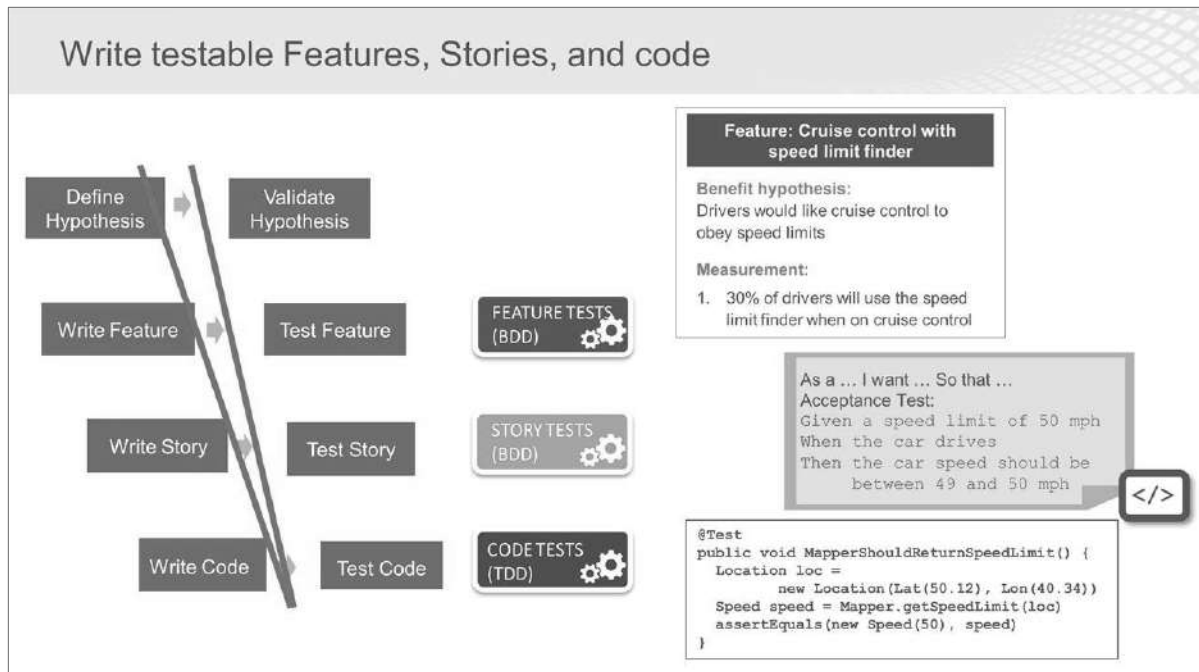
Notes:

Shift testing left for fast and continuous feedback



Notes:

5.1 Shift testing left



Notes:

Tests and requirements are related

- ▶ Every test is a requirement
 - If you cannot deploy with a test that fails, the test is a requirement that the system must meet
 - It is more effective to have requirements before creating an implementation
- ▶ Every requirement should have a test for it
 - Every passing test is a specification of how the system works

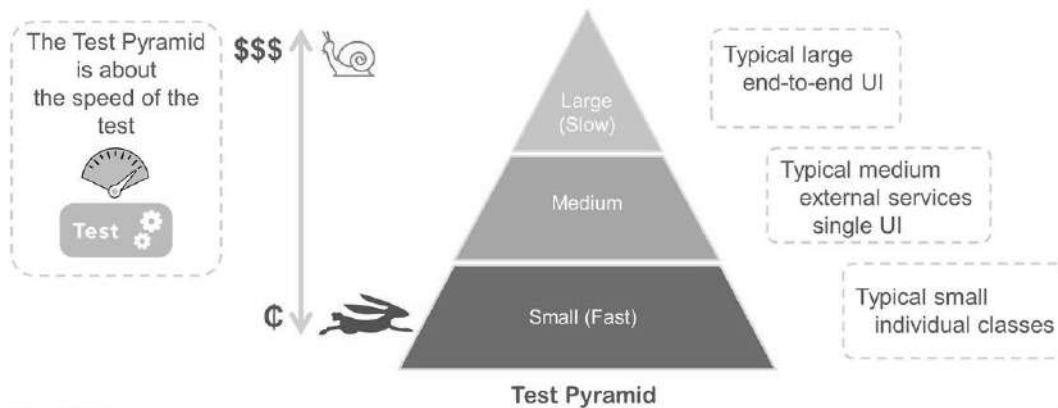
SCALED AGILE™ © Scaled Agile, Inc.

10

Notes:

Test-First naturally creates a pyramid of tests

- The Test Pyramid advocates a balanced portfolio of tests with many small, low-level, automated tests and fewer large, manual tests



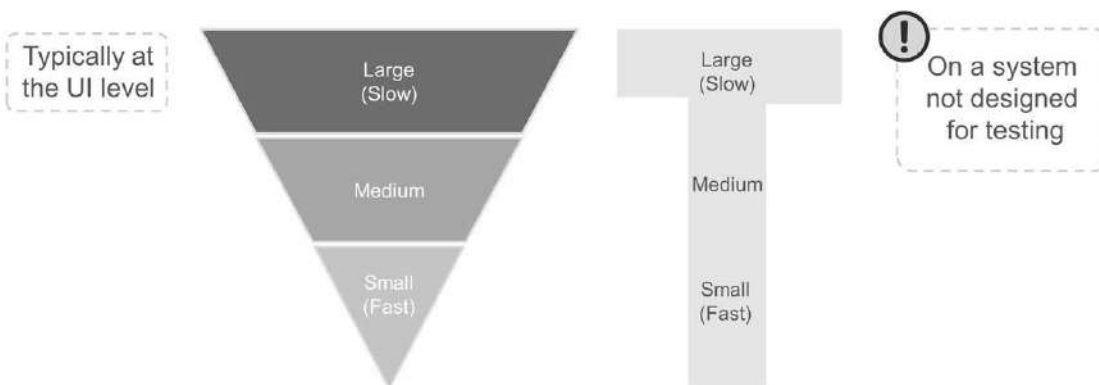
SCALED AGILE® © Scaled Agile, Inc.

11

Notes:

An inverted Test Pyramid is a test strategy anti-pattern

It slows development, delays feedback and encourages larger batches.



Do your tests look more like a pyramid or an inverted pyramid?

SCALED AGILE® © Scaled Agile, Inc.

12

Notes:


5.2 Explain the Agile testing matrix

SCALED AGILE © Scaled Agile, Inc.

13


Notes:

5.2 Explain the Agile testing matrix



Activity: What types of tests?

Share
3 min



We perform the following tests...

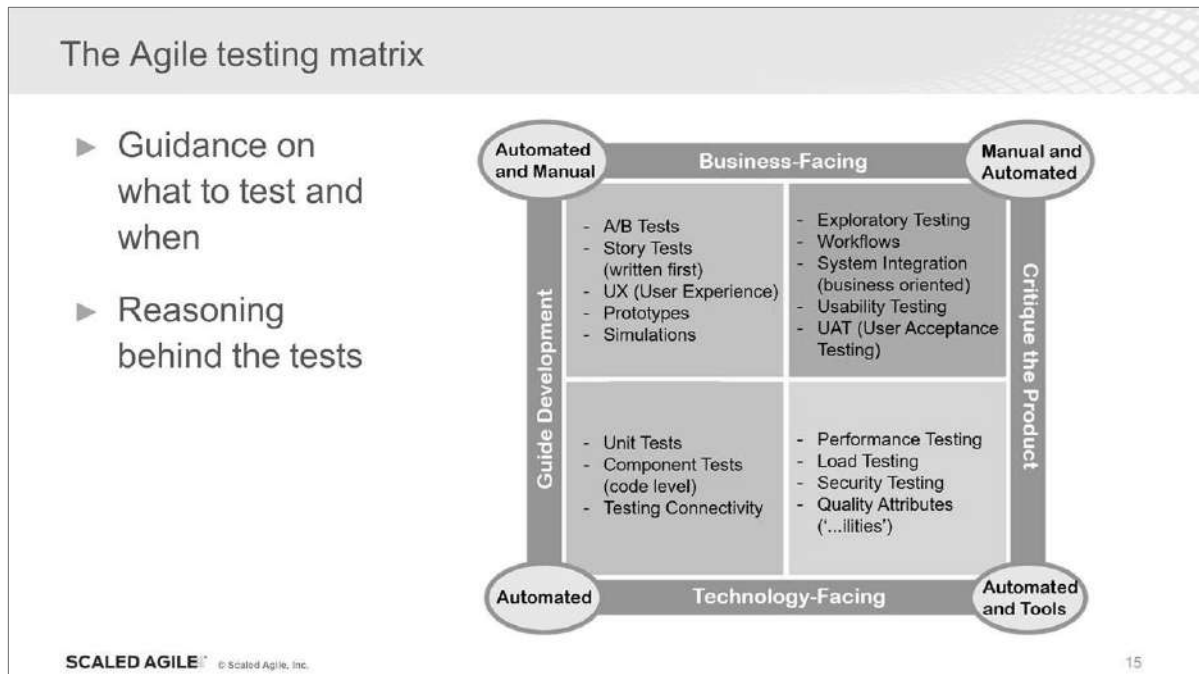
What types of tests do you currently perform at your organization?

SCALED AGILE® © Scaled Agile, Inc.

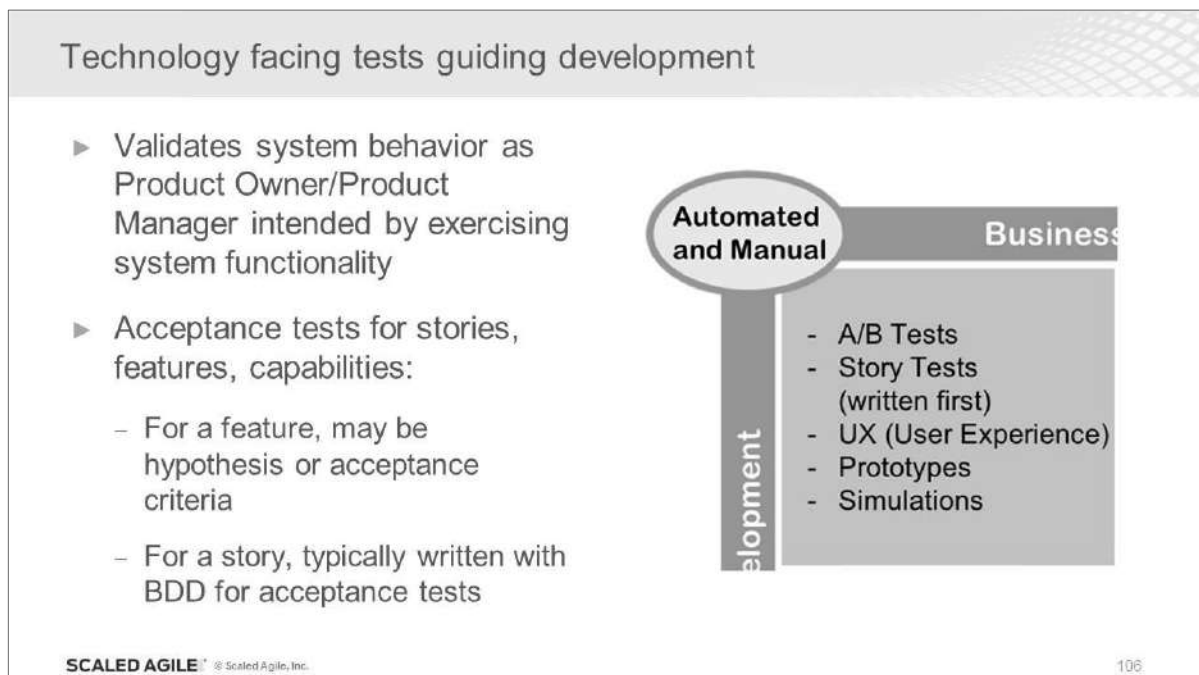
14

Notes:

5.2 Explain the Agile testing matrix



Notes:

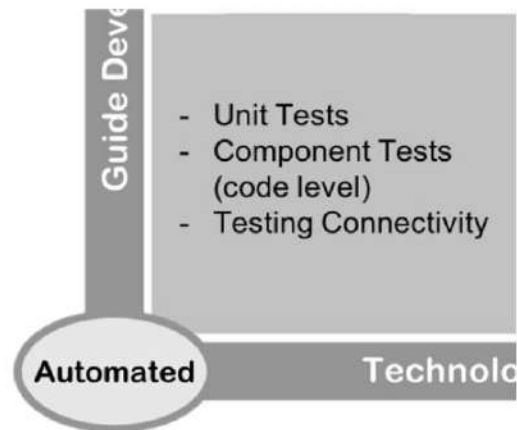


Notes:

5.2 Explain the Agile testing matrix

Technology facing tests guiding development

- ▶ Unit tests of code
- ▶ Component tests (modules, services, etc..)



SCALED AGILE® © Scaled Agile, Inc.

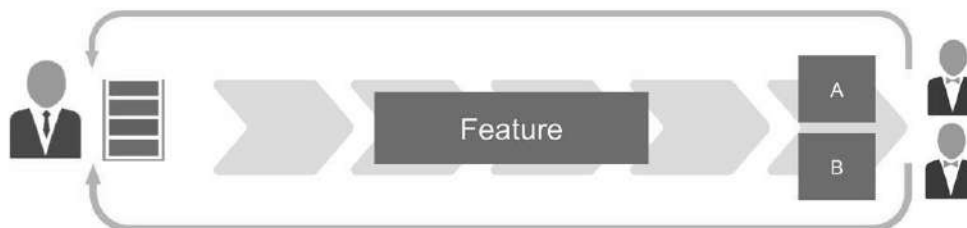
107

Notes:

Business-facing tests for validating hypotheses

- ▶ Hypotheses must be validated in production environment
- ▶ Constrain collateral damage from experimentation

A/B Testing: Use control groups to evaluate results against the hypothesis



SCALED AGILE® © Scaled Agile, Inc.

17

Notes:

5.2 Explain the Agile testing matrix

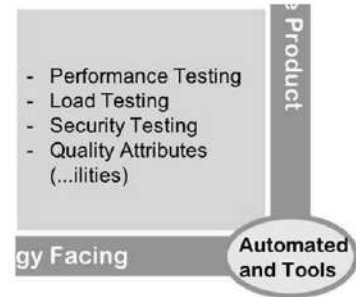
Technology-facing critiquing the Solution/environment

- ▶ Ensure system qualities and nonfunctional requirements (NFRs)

Canary Release: Slowly roll out changes to small subsets of users to validate assumptions before impacting everyone

Dark Launches Test a new feature in production before making it available to end users; useful for testing the impact of NFRs under load

NFR tests are automated through tools: load, performance, security, etc.



SCALED AGILE[®] © Scaled Agile, Inc.

19

Notes:

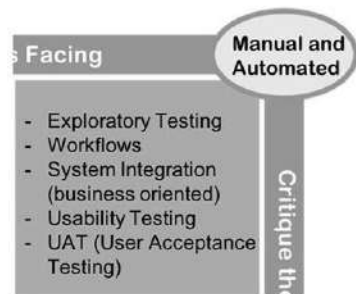
Business-facing critiquing the Solution

- ▶ Exercise workflow across many Stories' acceptance tests
- ▶ Include end user acceptance tests

Alpha: Defect detection from real users (typically internal)

Beta: Customer satisfaction and release readiness from users (external)

Mostly manual, some automated



SCALED AGILE[®] © Scaled Agile, Inc.

20

Notes:



Discussion: Non-scripted testing

Discuss



- What testing do you do that is not scripted or not test-planned ahead of time?



SCALED AGILE® © Scaled Agile, Inc.

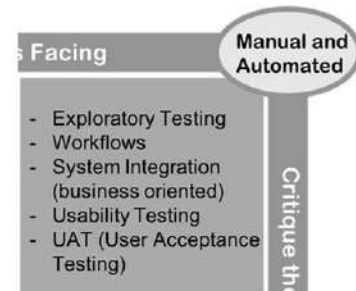
21

Notes:

Exploratory tests take unexpected paths through the system

- ▶ Exercise things that are possible but not 'expected'
 - Does the system respond properly to unexpected input?
- ▶ Have we defined how the system is supposed to respond?
 - Unexpected paths are tests, and tests are requirements
 - Define behavior prior to implementation as tests for developers

Typically manual, but may become automated



SCALED AGILE® © Scaled Agile, Inc.

22

Notes:

Possible exploratory tests for cruise control with speed limit

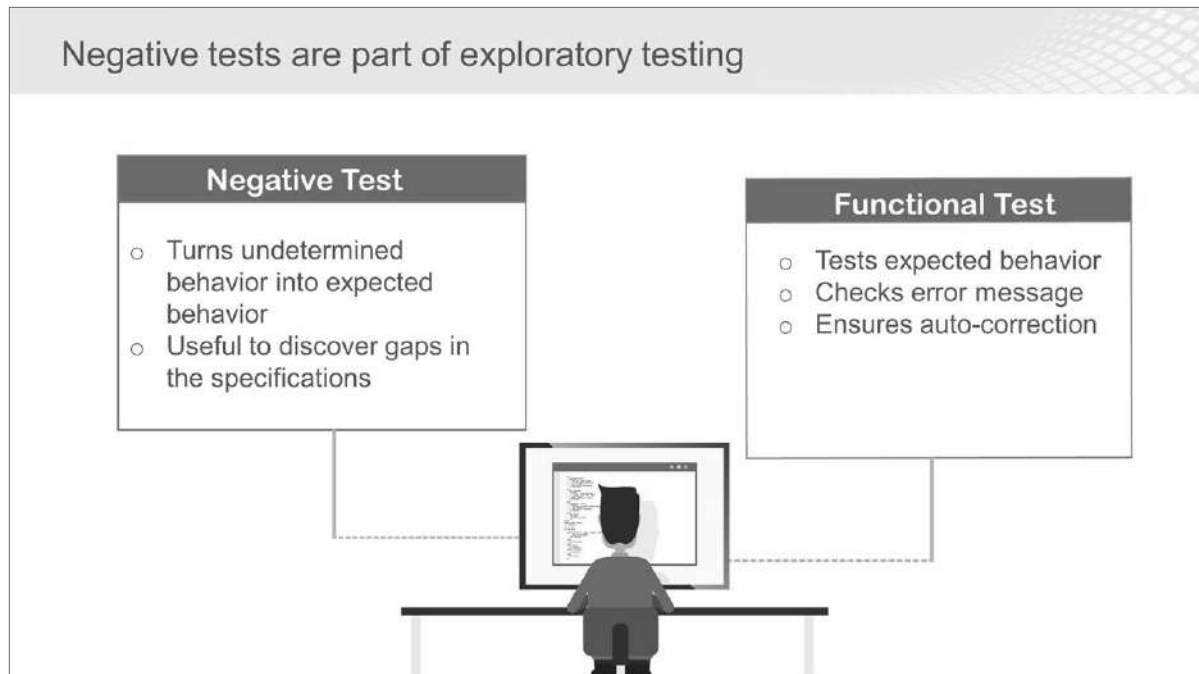
Examples:

1. Create a road with speed limits that vary greatly and see if speed limits are followed
2. Create a road with no speed limits given and see what happens
3. Create a road with speed limit changes that would be impossible to follow and see what happens




How should the system behave with 'impossible' tests (#3)?

Notes:



Notes:

5.2 Explain the Agile testing matrix



Discussion: Testing and experimentation in production

Discuss
3 min

Share
2 min

► **Step 1:** At your table, discuss:

- What can you learn in production, both functionally (MMF) and technically, that you cannot learn elsewhere?
- What are the risks of testing in production vs. not testing in production?

► **Step 2:** Considering that a risk is a function of the probability of an error and the impact if the error occurs, discuss:

- What are ways we can reduce probability or severity?

I don't test often...but when I do, I test in production.
—Google TechTalk on Testing at Netflix

SCALED AGILE® © Scaled Agile, Inc.

25

Notes:


5.3 Describe the role of nonfunctional requirements (NFRs)

SCALED AGILE © Scaled Agile, Inc.

26


Notes:

5.3 Describe the role of nonfunctional requirements (NFRs)



Discussion: What types of NFRs do you consider?

Share
4 min



We perform these NFR tests manually and these automatically ...

What NFR tests do you currently perform at your organization?

SCALED Agile, Inc.

27

Notes:

Nonfunctional requirements (NFRs)

- ▶ NFRs are also called system qualities or cross-functional requirements.
- ▶ Typically they are specified attributes of a solution.
- ▶ Categories include:
 - Usability
 - Reliability
 - Security
 - Performance



SCALED AGILE® © Scaled Agile, Inc.

28

Notes:

Testing NFRs

One way to approach NFR testing:

Step 1:

- ▶ Name: In the form Quality. SubQuality
- ▶ Scale: What to measure (units)
- ▶ Meter: How to measure (method)

Step 2:

- ▶ Target: Success level to achieve
- ▶ Constraint: Failure level to avoid
- ▶ Baseline: Current level

SCALED AGILE® © Scaled Agile, Inc.

29

Notes:

Example: NFR test for automated speed limit detection

- ▶ **Name:** *Usability.Efficiency*
- ▶ **Scale:** Number of times the users decides to set the speed manually
- ▶ **Meter:** Average observed results per trip from monitoring

Constraint:
.15 time per
mile traveled

Baseline:
.1 times per
mile traveled

Target:
.01 times per
mile traveled



SCALED AGILE® © Scaled Agile, Inc.

30

Notes:

5.3 Describe the role of nonfunctional requirements (NFRs)



Activity: Nonfunctional requirements (NFR) tests

Prepare
5 min

Share
5 min

- **Step 1:** Get together with your team
- **Step 2:** Considering your own context or the *Autonomous Vehicle Parking* domain, write a nonfunctional requirement (NFR) test with a *constraint/baseline/target*

Constraint

Baseline

Target



SCALED AGILE® © Scaled Agile, Inc.

31

Notes:


5.4 Build quality in throughout the pipeline

SCALED AGILE © Scaled Agile, Inc.

32

Notes:

5.4 Build quality in throughout the pipeline



Activity: Your current pipeline

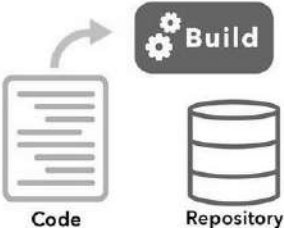
Prepare
5 min

Share
2 min

► **Step 1:** Individually in your workbook or as a team, list:

- The steps it currently takes for the build process
- The steps it takes to deploy
- The step at which source code 'final' commit (commit or merge to main branch) occurs

► **Step 2:** Be prepared to share with the class

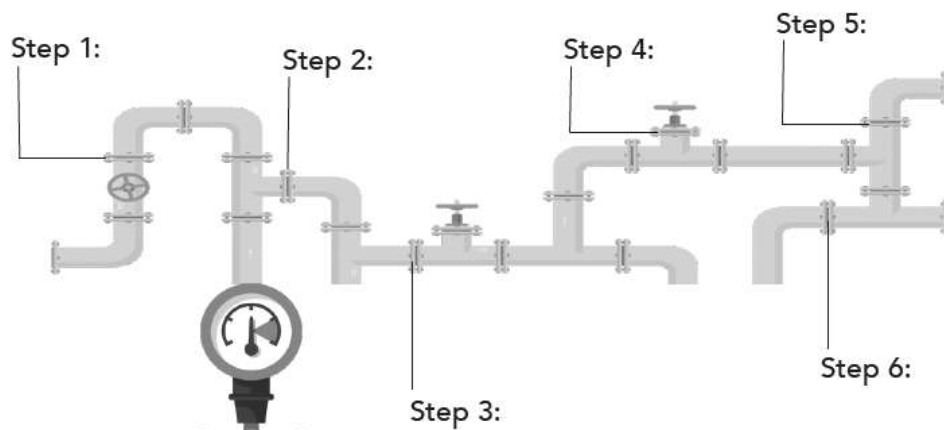


Code Build Repository

SCALED AGILE® © Scaled Agile, Inc.

33

Notes:



5.4 Build quality in throughout the pipeline

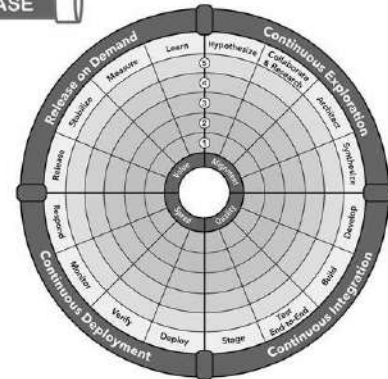
SAFe's Continuous Delivery pipeline and DevOps Capabilities

Workflows, activities, and automation needed to provide continuous release of value.



Four elements of the pipeline:

- ▶ Continuous Exploration
- ▶ Continuous Integration
- ▶ Continuous Deployment
- ▶ Release on Demand



SCALED AGILE® © Scaled Agile, Inc.

34

Notes:

Intersection of Agile software engineering and DevOps



Agile software engineering focus:

- ▶ Write testable Stories
- ▶ Develop small changes with tests
- ▶ Build small changes
- ▶ Test changes in development
- ▶ Test continuously in staging
- ▶ Deploy for further testing

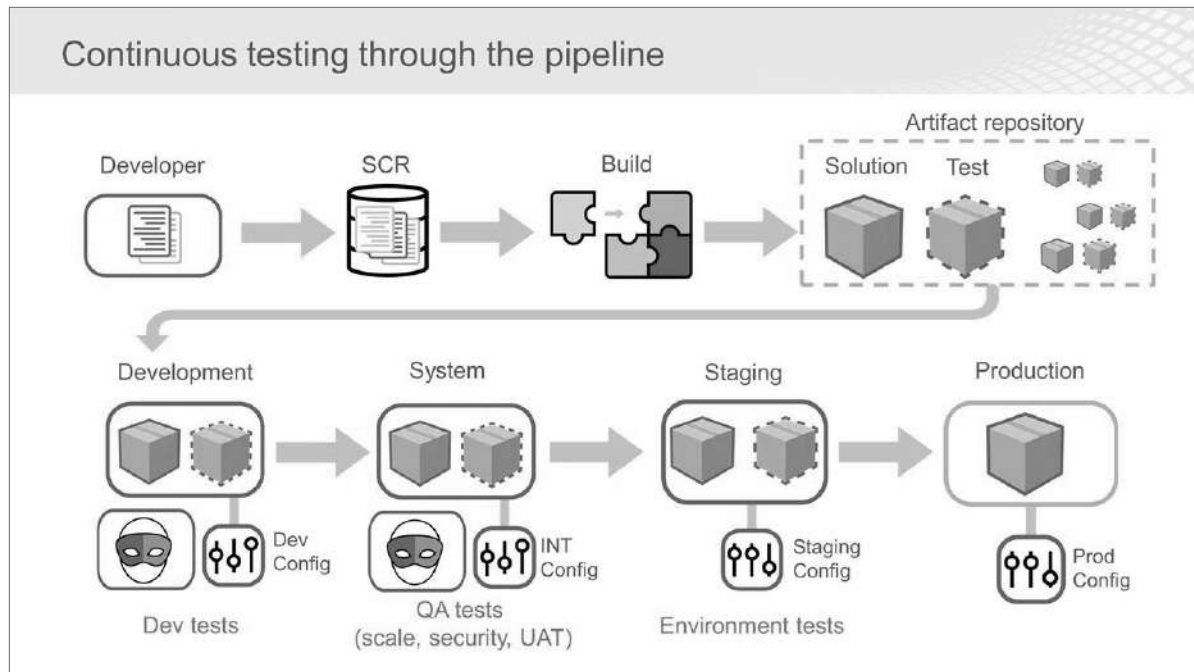


SCALED AGILE® © Scaled Agile, Inc.

35

Notes:

5.4 Build quality in throughout the pipeline



Notes:

What is a package?

- **Package:**
 - Individually deployable component such as a microservice to a container
- **Build once, deploy anywhere**
 - Packages deployed to each environment are the same
 - Configuration may differ for each environment, such as with setup for test doubles

App Package **Test Package**

The diagram shows two packages: an **App Package** (represented by a solid cube) and a **Test Package** (represented by a dashed cube).

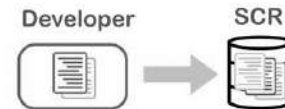
SCALED AGILE © Scaled Agile, Inc.

Notes:

5.4 Build quality in throughout the pipeline

Example developer flow for a desktop application

1. Load necessary source on local developer machine with libraries for rest of system
2. Have developer test for current change (unit tests plus applicable BDD tests)
3. Modify code until tests pass
4. When tests pass, run test for entire app on local developer machine
5. When they pass, refresh or merge source locally on developer machine
6. Run all unit tests plus applicable customer focused tests locally
7. When they pass, check in change to SCR



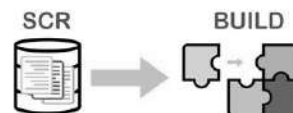
SCALED AGILE[®] © Scaled Agile, Inc.

38

Notes:

Continuous flow for builds

- ▶ SCR:
 - On every check-in, perform build
- ▶ BUILD:
 - Perform static code analysis: common errors, potential security issues, code style, redundancy violations
 - Build all relevant artifacts
 - Run all unit tests and applicable customer focused tests (smoke test)
 - If all pass, deploy to developer test environment



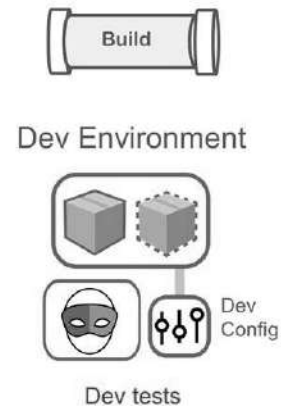
SCALED AGILE[®] © Scaled Agile, Inc.

39

Notes:

Continuous Integration flow for testing

- ▶ Developer build environment (fast feedback – ‘smoke test’):
 - Use test doubles where needed
 - Run BDD tests
 - Run other tests that can be run rapidly
 - If they pass, deploy to developer test environment
- ▶ Developer test environment (slower feedback):
 - Run other tests (BDD, performance, etc.) that can be run automatically and reasonably quickly (e.g. overnight or few hours)
 - If pass, deploy to QA test environment (or deploy to QA test/NFR/user environment in parallel)



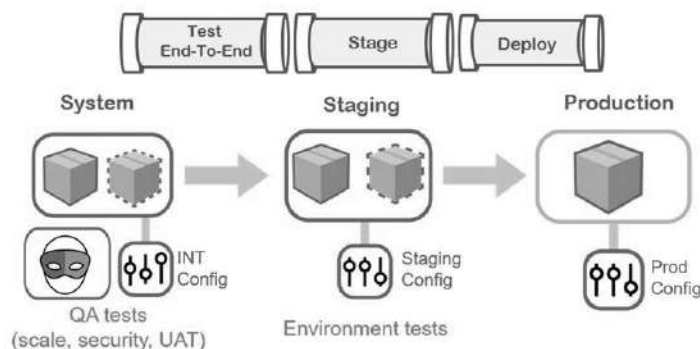
SCALED AGILE® © Scaled Agile, Inc.

40

Notes:

Continuous test into production

- ▶ Ensure test environments match production as much as possible
- ▶ Maintain all configurations in version control



SCALED AGILE® © Scaled Agile, Inc.

41

Notes:



Agile Software Engineering Action Plan



- **Step 1:** Locate the Agile Software Engineer Action Plan at the end of your workbook
- **Step 2:** Considering what we have covered, write down at least one improvement item in your Agile Software Engineer Action Plan
- **Step 3:** Share the item you wrote with the class.



Notes:



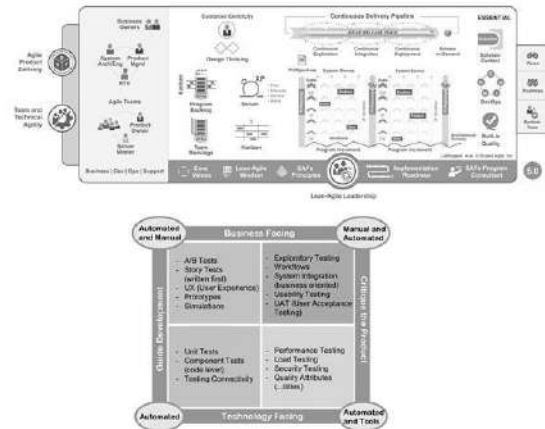
Discussion: Thinking Test-First in SAFe

Share



► Discuss:

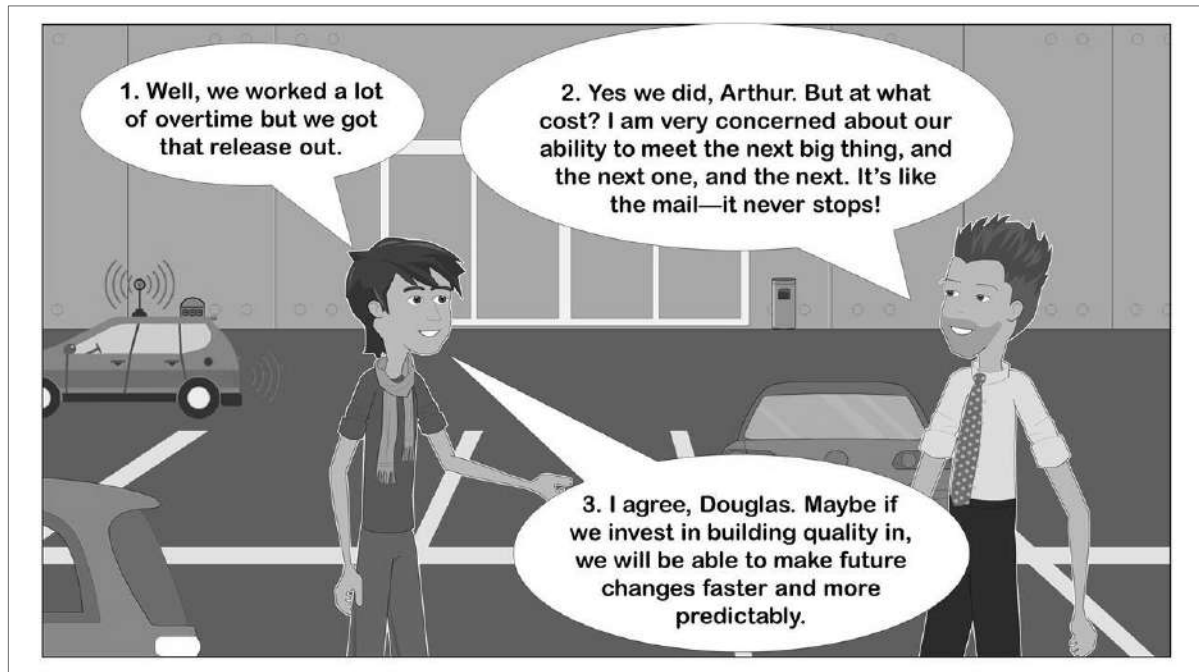
- Where in SAFe does shift testing left apply?
- Who in SAFe would be responsible for tests in each of the quadrants?



SCALED AGILE® © Scaled Agile, Inc.

43

Notes:



Notes:

Lesson review

In this lesson, you:

- ▶ Shifted testing left
- ▶ Explained the Agile testing matrix
- ▶ Described the role of nonfunctional requirements (NFRs)
- ▶ Built quality in throughout the pipeline

Notes:

Lesson 6

Discovering Story Details

Learning Objectives:

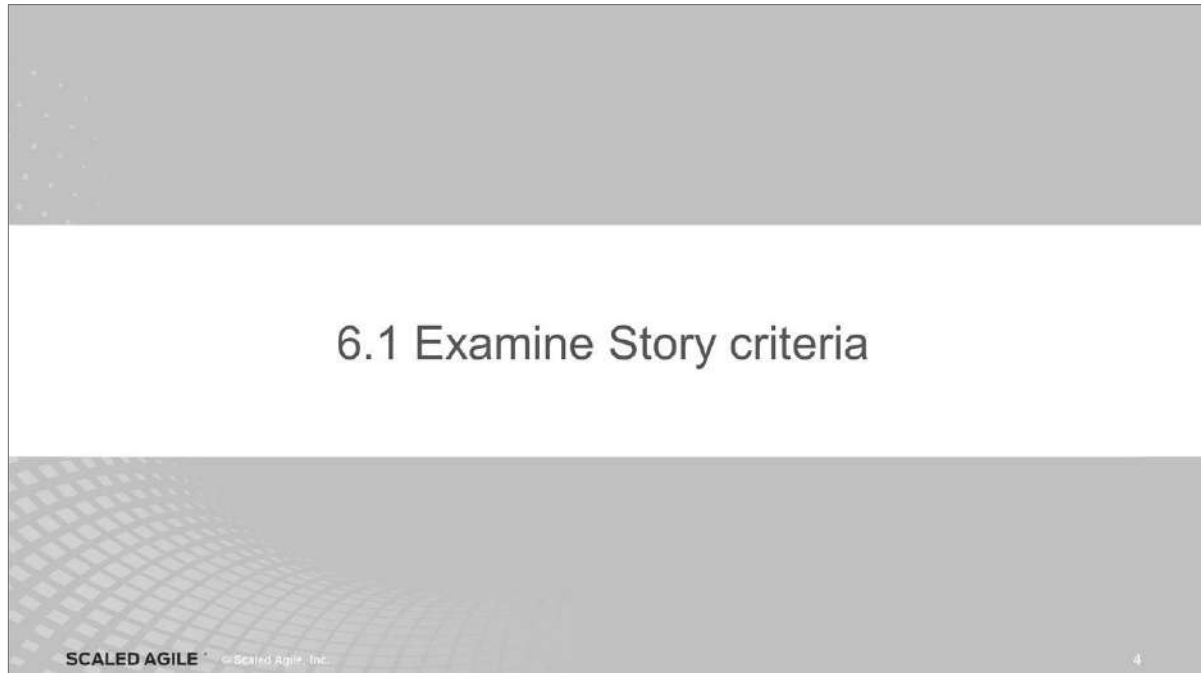
- 6.1 Examine Story criteria
- 6.2 Split stories to reduce the minimum marketable feature (MMF)
- 6.3 Create workflow or story maps
- 6.4 Identify assumptions and risks




SAFe® Authorized Course Attending this course gives students access to the SAFe® Agile Software Engineer exam and related preparation materials.




Notes:




Notes:



Activity: What are the INVEST criteria





**'I' in 'INVEST
stands for...**

**What are the INVEST
criteria for a story?**

SCALED AGILE © Scaled Agile, Inc.

5

Notes:

INVEST in a good Story

- ▶ Write Stories that can be developed separately
- ▶ Write Stories in which scope can be negotiated
- ▶ Write Stories that are valuable to the Customer
- ▶ Write Stories that can be estimated
- ▶ Write Stories that can fit in an Iteration
- ▶ Write Stories that are testable

I	<u>I</u> ndependent
N	<u>N</u> egotiable
V	<u>V</u> aluable
E	<u>E</u> stimable
S	<u>S</u> mall
T	<u>T</u> estable

SCALED AGILE® © Scaled Agile, Inc.

139

Notes:

Definition of scenario

DEFINITION

- ▶ Defines one behavior of the system
 - A flow (or workflow) that a user performs
 - Typically from the external view of the system
- ▶ One format for describing scenario
 - Given current state
 - When action or event occurs
 - Then state change or output
- ▶ Relationship between story and scenario: multiple scenarios to story

SCALED AGILE® © Scaled Agile, Inc.

7

Notes:

Acceptance criteria and scenarios

- ▶ Acceptance criteria define acceptable behavior for a Story's done state
- ▶ Criteria written as scenarios may provide additional detail

Set cruise control
Acceptance criteria:
• The vehicle's speed must be close to the set speed.

Template for Scenario

Given current state
When action or event occurs
Then state change or output

Scenario

Given car is moving
When speed is set
Then speed is close to set speed

Notes:

Stories have many scenarios

Scenario going uphill

Given car is moving uphill
When speed is set
Then speed is as close as possible to set speed

Scenario going downhill

Given car is moving downhill
When speed is set
Then speed will never go above the set speed

Notes:

Use cases are scenarios

- ▶ Use cases have:
 - Pre-conditions: what must be true before case executes
 - Main flow: what happens
 - Post-conditions: what is true afterwards
- ▶ Main flow is one *When*
 - Exceptions and alternate flows are other *Whens* in separate
Given / When / Then



If you are already using use cases

SCALED AGILE[®] © Scaled Agile, Inc.

10

Notes:

Use case example of a scenario/flow

Name: Accelerate to speed.

Actor: Driver.

Pre-condition: Car is not moving. Engine is on. Driver is in seat.

Post-condition: Car is moving at least 30 mph.

Main flow:

- Driver presses down the gas pedal.
- Five seconds pass.



What exceptions are there? These become different scenarios.

SCALED AGILE[®] © Scaled Agile, Inc.

11

Notes:

Other terms for these scenario/flow concepts

Initial State	Given	Setup	Arrange	Assemble	Pre-conditions
Action	When	Trigger	Act	Activate	Main course/ exceptions/ alternatives
Final State and/or Output	Then	Verify	Assert	Assert	Post-conditions

SCALED AGILE® © Scaled Agile, Inc.

12

Notes:

Using personas to better understand users

Personas are detailed fictional characters acting as a representative user



Jane

- Law-abiding driver
- Obeys all traffic signs
- Wants to save on gas

Mileage-sensitive

As Jane, I want to travel at the legal limit and operate in an energy saving manner so that I do not get a ticket and I save money



Bob

- Impatient driver
- Ignores traffic signs if they slow him down


Time-sensitive

As Bob, I want to travel at the maximum speed the roadway and my vehicle safely allows so that I arrive quickly

SCALED AGILE® © Scaled Agile, Inc.

13

Notes:

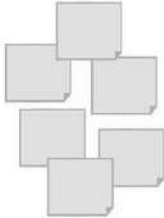


Activity: Writing Stories using INVEST and personas

Prepare
15 min

Share
5 min

- ▶ **Step 1:** In your team, brainstorm three or more stories for your MMF you identified previously.
 - Define and use personas; for the *Autonomous Vehicle Parking* domain, you may use the personas of Jane and Bob
- ▶ **Step 2:** Choose two of the Stories.
 - Write acceptance criteria for those stories in **Given/When/Then** form
- ▶ **Step 3:** Pair with someone from another team and share your results.



SCALED AGILE® © Scaled Agile, Inc.

14

Notes:



Video: Estimation and Fence Posts

Duration
1 min




<https://vimeo.com/374272836/0cd80e4a86>

SCALED AGILE® © Scaled Agile, Inc.

148


Notes:

Summary of Agile estimation techniques




Estimating Poker

- Common, well-known
- Useful for small number of items and participants



White Elephant

- Works with large number of items and participants
- Ensures everyone participates equally



Relative to Prior Features/Stories

- Similar to white elephant estimation, with previous Iteration's Stories placed on board relative to each other

SCALED AGILE® © Scaled Agile, Inc.

16

Notes:

Transforming acceptance criteria into tests

► Acceptance criteria:

- Define acceptable behavior

```
Given car is moving
When speed is set
Then speed is close to set speed
```

► Acceptance tests:

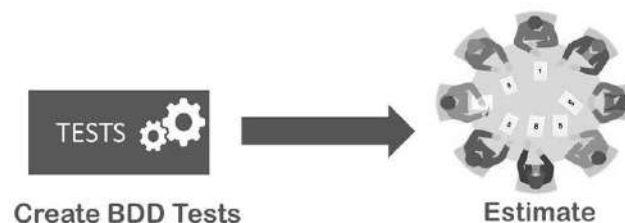
- Are derived from acceptance criteria
- Define specific pass/fail behavior
- Will be covered more in the next lesson

```
Given car is moving at 10 mph
When speed is set to 30 mph
Then car is at 29 mph in less
than 5 seconds
And car speed is no lower than 29
mph and no higher than 31 mph
```

Notes:

Use BDD to improve estimates

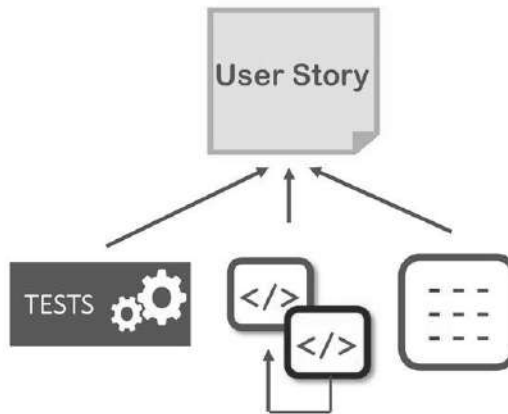
- Writing tests better clarifies Stories
- The number and complexity of tests can be used as a heuristic for estimation
- Create tests when you want a better estimate




Notes:

Consider **all** work in estimates


- For each Story, factor in the time to:
 - Write and automate tests
 - Refactor code in order to maintain quality
 - Write tests for code that does not have tests




Notes:



Discussion: Definition of done





Consider that Story descriptions should include a definition of done.

What is included in your definition of done ?

Would creating tests be part of the definition of done, or part of your estimates?

SCALED AGILE © Scaled Agile, Inc.

20

Notes:


6.2 Split Stories to reduce the minimum marketable feature (MMF)

SCALED AGILE © Scaled Agile, Inc.


21


Notes:

6.2 Split stories to reduce the minimum marketable feature (MMF)



Activity: How have you split Stories in the past?





We split Stories using...

How have you split Stories in the past?

What reasons did you have and what techniques have you used?

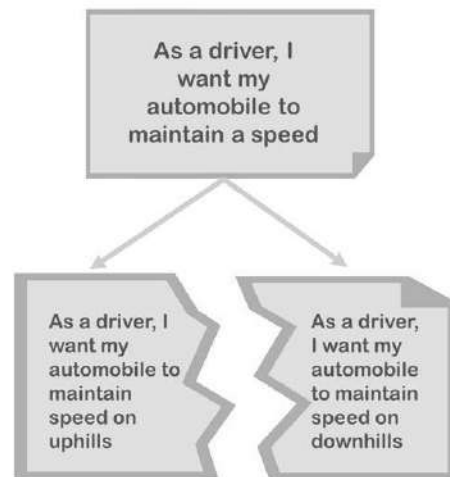
SCALED AGILE® © Scaled Agile, Inc.

22

Notes:

Why split Stories?

- ▶ In support of MMF, decrease size to minimum:
 - Split Stories into essential and non-essential parts and eliminate non-essential
 - Ensure you have something releasable
- ▶ In support of feedback:
 - Deploy small Stories to get technical/user feedback quickly (maximize feedback)
- ▶ In support of iteration planning:
 - Split Stories so they fit into an iteration



SCALED AGILE® © Scaled Agile, Inc.

23

Notes:

Apply some common Story-splitting techniques

Splitting techniques:

- ▶ Business rule variations (e.g. single variation, then remainder)
- ▶ Workflow steps (for multi-step stories)
- ▶ Simple/complex (e.g. search for single word, then for phrases)
- ▶ Scenarios (e.g. use case exceptions)



SCALED AGILE® © Scaled Agile, Inc.

24

Notes:

Split stories by their acceptance criteria

As a driver, I
want to never
exceed the speed
limit

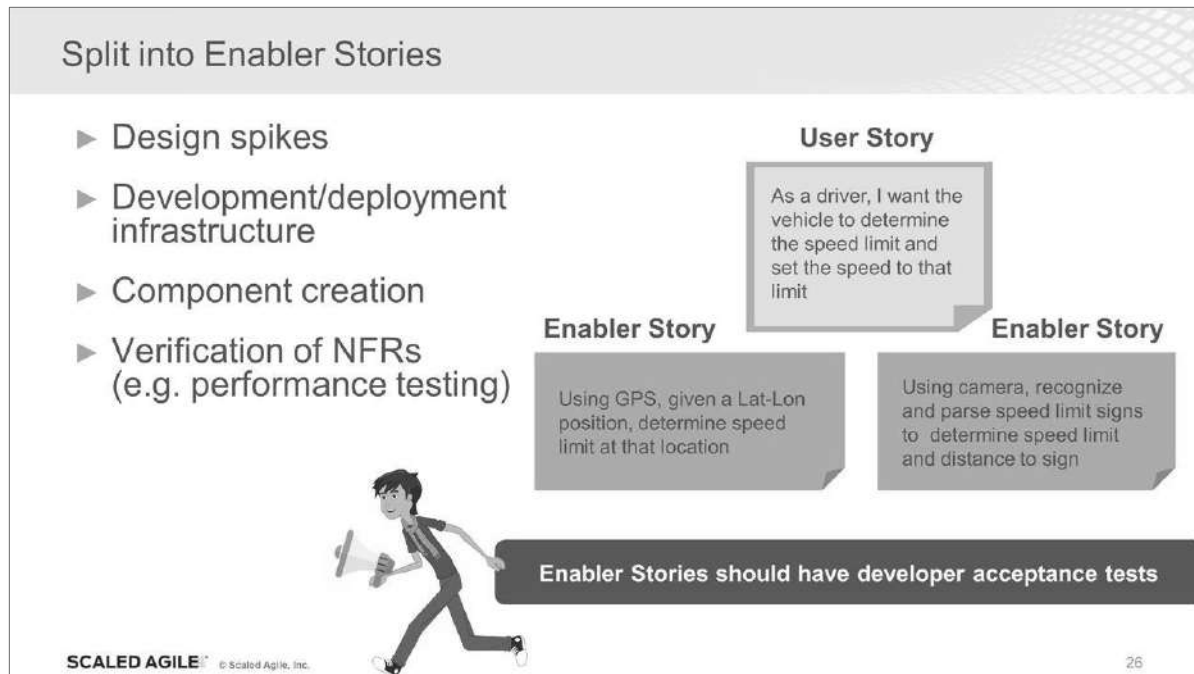
Given I am driving at the current speed limit
When a new speed limit is lower
Then current speed becomes new speed before it applies

Given I am driving at the current speed limit
When a new speed limit is higher
Then current speed becomes new speed after it applies

SCALED AGILE® © Scaled Agile, Inc.

25

Notes:



Notes:

6.2 Split stories to reduce the minimum marketable feature (MMF)



Activity: Splitting Stories

Prepare
5 min

Share
5 min

- ▶ **Step 1:** With your team, review the Stories you created
- ▶ **Step 2:** Consider the Stories for potential splitting based on the criteria discussed previously
 - How could splitting create a smaller MMF or help release an MMF faster?



SCALED AGILE © Scaled Agile, Inc.

27

Notes:

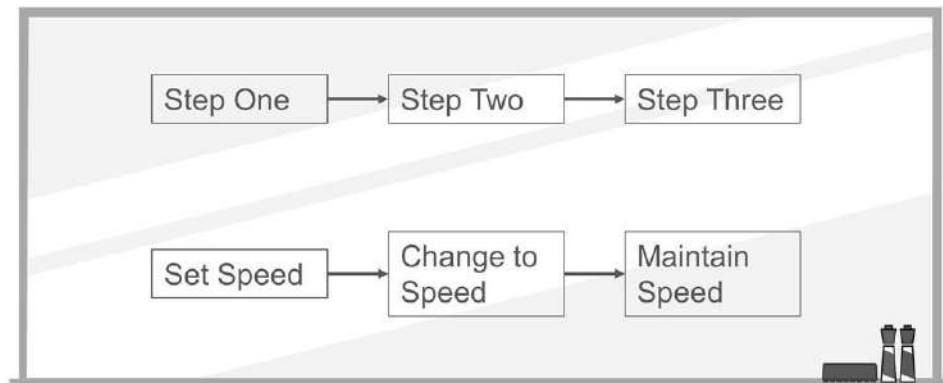
6.3 Create workflow or Story maps

SCALED AGILE® © Scaled Agile, Inc.

28

Notes:

Decompose Features into workflow steps



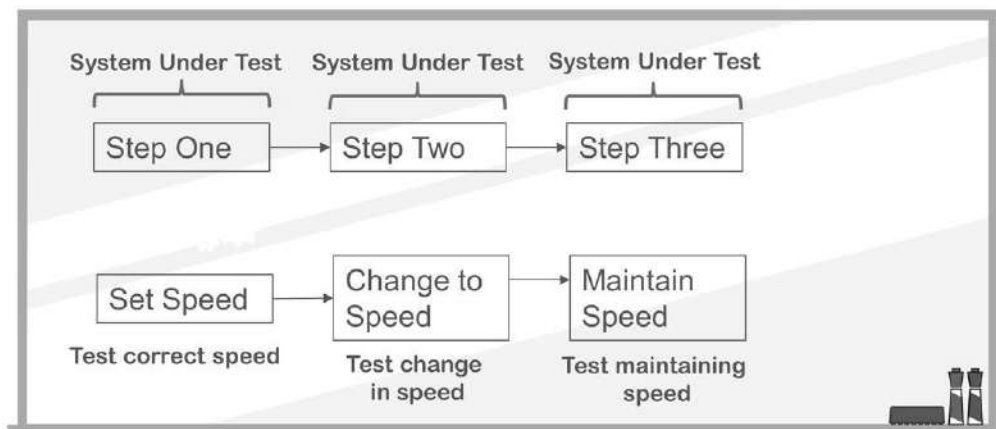
SCALED AGILE® © Scaled Agile, Inc.

29

Notes:

6.3 Create workflow or story maps

Workflows steps may have their own acceptance criteria



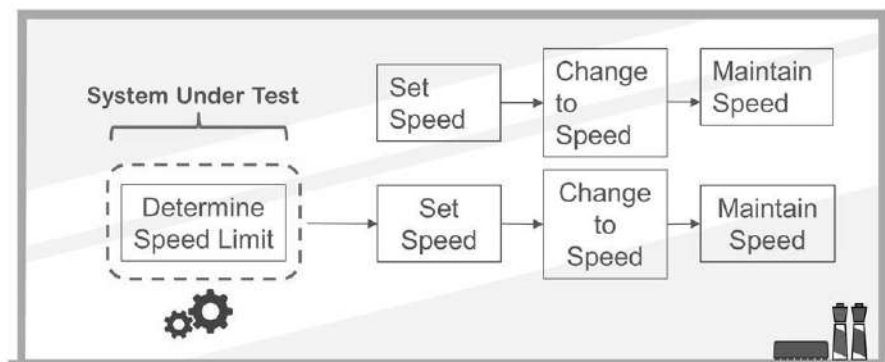
SCALED AGILESM © Scaled Agile, Inc.

30

Notes:

New Stories add a step or alter an existing step in a workflow

Changes to workflow can add more acceptance criteria.



SCALED AGILESM © Scaled Agile, Inc.

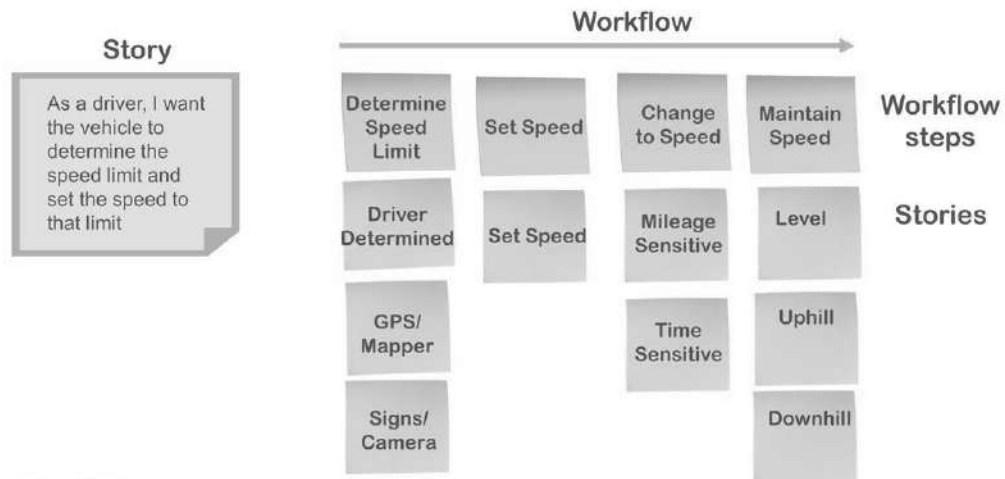
31

Notes:

6.3 Create workflow or story maps

Story maps show workflow steps and alternatives

Show all the scenarios that make up a overall Feature or Story.

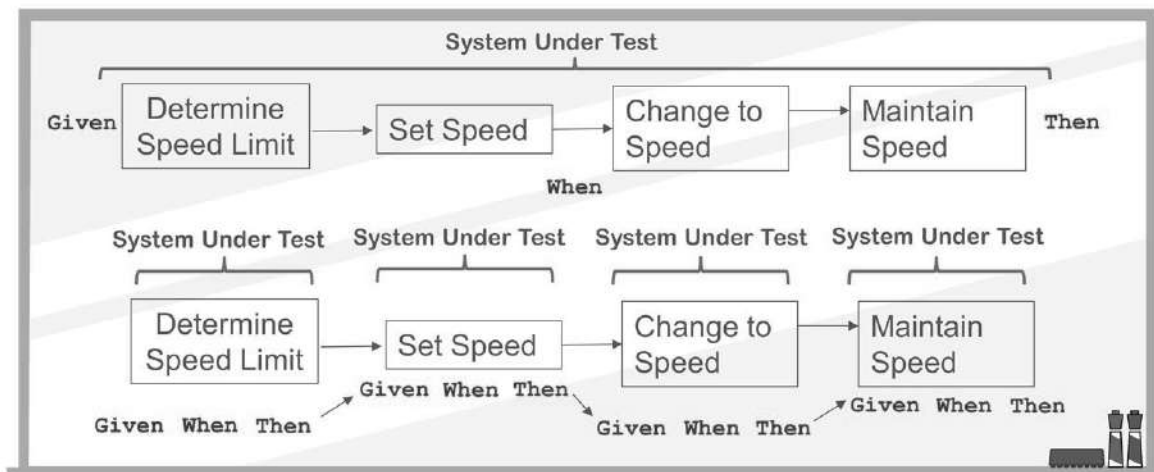


SCALED AGILE[®] © Scaled Agile, Inc.

32

Notes:

Test both end-to-end workflow and individual steps




SCALED AGILE[®] © Scaled Agile, Inc.

33

Notes:

6.3 Create workflow or story maps



Activity: Workflow model

Prepare


10 min

Share

5 min

- **Step 1:** With your team, using a flip chart sheet, either break a Story into workflow steps or create a Story map for multiple Stories
- **Step 2:** Discuss how would you break down testing based on the workflow steps

Workflow



```
graph LR; A[Determine Speed Limit] --> B[Set Speed]; B --> C[Change to Speed]; C --> D[Maintain Speed];
```

SCALED AGILE® © Scaled Agile, Inc.

34

Notes:

6.4 Identify assumptions and risks

SCALED AGILE © Scaled Agile, Inc.

35

Notes:

What are assumptions?

- ▶ Assumptions are: Things you assume to be true



Examples:

- Assume a car has sufficient power to maintain a speed
 - Assume a car has sufficient energy storage to maintain speed for a distance
- ▶ Document assumptions make implicit assumptions explicit: they are what you want to explore in testing
 - ▶ The more assumptions you check, the more scenarios are checked by the system

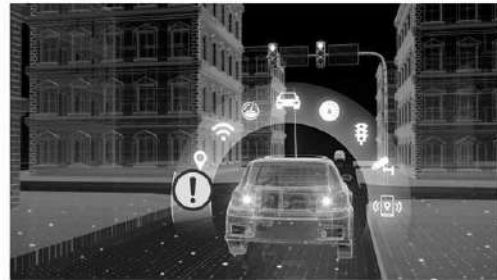
SCALED AGILE © Scaled Agile, Inc.

36

Notes:

What are risks?

- ▶ Risks are a combination of severity/impact and probability
 - Manage risks to decrease either or both
 - List risks so they can be managed
- ▶ Risks in implementation and deployment
 - Camera gets dirty so it cannot detect signs
 - GPS satellites crash so there is no signal
 - No GPS data in a location (e.g. in a tunnel)



SCALED AGILE® © Scaled Agile, Inc.

37

Notes:



Discussion: Assumptions and risks

Discuss
4 min

Share
2 min

► **Step1:** As a team, discuss the following:

- What assumptions have you made in your Story? List a few of them.
- What risks does your Story have? List a few of them.

► **Step 2:** Share with the class.



SCALED AGILE® © Scaled Agile, Inc.

38

Notes:

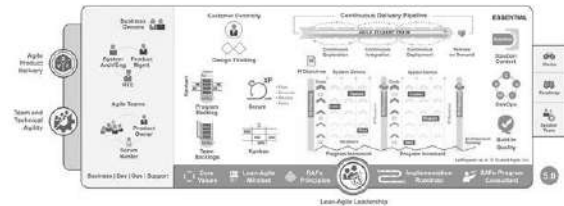


Discussion: Creating MMF in SAFe

Discuss



- ▶ In SAFe, which roles are responsible for creating the MMF?
- ▶ Which SAFe roles decompose the MMF into Stories—applying INVEST, personas, estimating, and splitting?
- ▶ When in SAFe would MMFs be defined? What about Stories?



SCALED AGILE® © Scaled Agile, Inc.

40

Notes:



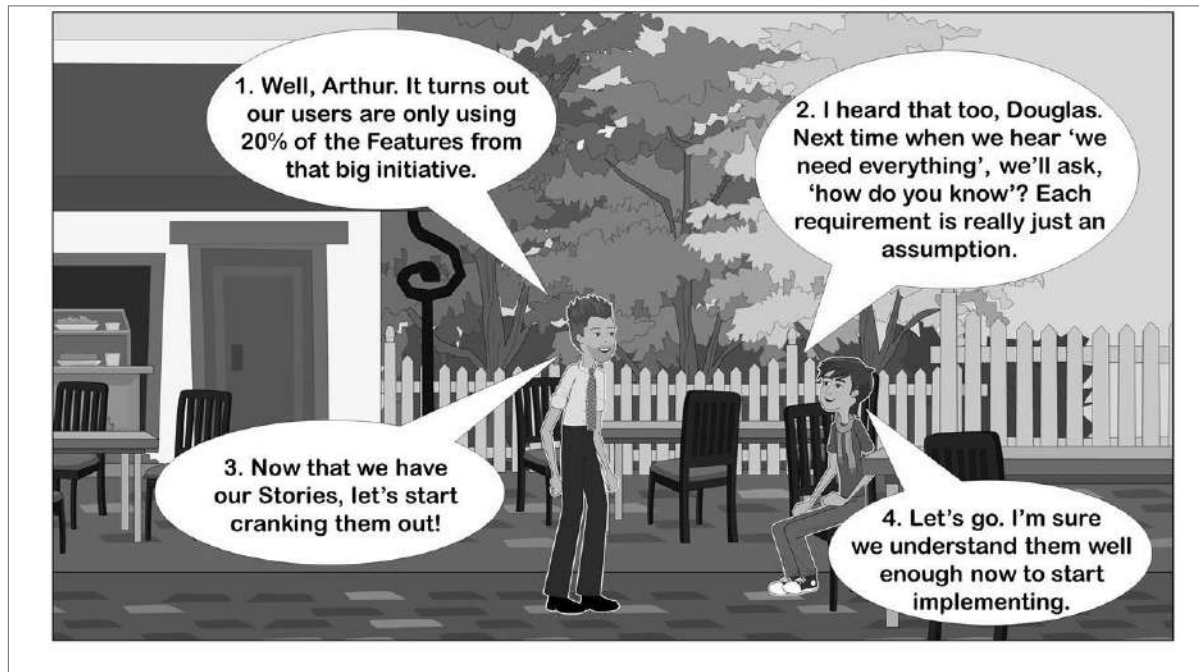
Agile Software Engineering Action Plan



- ▶ **Step 1:** Locate the Agile Software Engineer Action Plan at the end of your workbook
- ▶ **Step 2:** Considering what we have covered, write down at least one improvement item in your Agile Software Engineer Action Plan
- ▶ **Step 3:** Share the item you wrote with the class.



Notes:



Notes:

Lesson Review

In this lesson, you:

- 6.1 Examined Story criteria
- 6.2 Split Stories to reduce the MMF
- 6.3 Created workflow or Story maps
- 6.4 Identified assumptions and risks

Notes:

Lesson 7

Creating a Shared Understanding with Behavior-Driven Development (BDD)

Learning Objectives:

- 7.1 Apply Behavior-Driven Development (BDD) for shared understanding
- 7.2 Specify desired behavior for domain terms
- 7.3 Describe behavior for business rules and algorithms with tests
- 7.4 Test the user interface (UI)
- 7.5 Apply test doubles to BDD
- 7.6 Find existing tests impacted by new requirements



SAFe® Authorized Course Attending this course gives students access to the SAFe® Agile Software Engineer exam and related preparation materials.



Notes:

7.1 Apply Behavior-Driven Development (BDD) for shared understanding

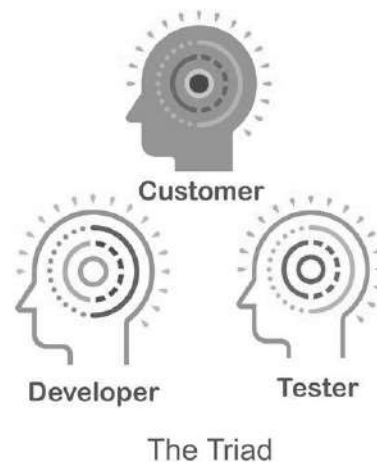
SCALED AGILE © Scaled Agile, Inc.

4

Notes:

Understanding behavior involves multiple perspectives

- ▶ A 'triad' of cognitively diverse thinking perspectives can facilitate better understanding.
- ▶ Customer thinking: Viable, desirable
- ▶ Developer thinking: Technically feasible
- ▶ Tester thinking: Exceptions, edge cases, boundary conditions

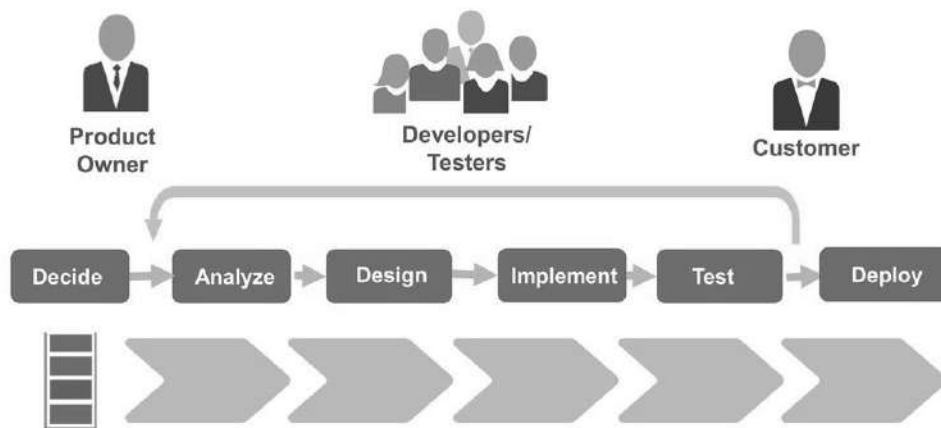


SCALED AGILE © Scaled Agile, Inc.

5

Notes:

The flow of behavior-driven development (BDD)



Notes:

Behavior-driven development: From ambiguity to precision

- ▶ Behavior is often first described in general terms, which can be ambiguous.
- ▶ Specific examples of behavior provide better understanding.
- ▶ The examples can directly become tests, or they can lead to specific behaviors which then are transformed into tests.
- ▶ One path is:



SCALED AGILE[®] © Scaled Agile, Inc.

7

Notes:

Acceptance criteria and tests: Variations of the same concept

► Acceptance criteria for a scenario

- More generic, like an outline

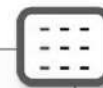
```
Given a speed limit
When the car drives
Then the car should be under the speed limit
```



► Example of a scenario, which can be an acceptance test

- Specific pass/fail, may uncover detail

```
Given a speed limit of 50 mph ●----- Setup
When the car drives ●----- Event
Then the car speed should be between 49 and 50 mph ●--- Outcome/Test
```



Notes:

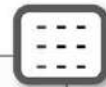
Difference between behavior and test of behavior

► Given, When, Then specifies desired behavior

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

► A test checks the Then of the behavior

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



Notes:

BDD and ATDD are in essence the same thing

- ▶ Both define acceptance tests and use Given/When/Then forms
- ▶ BDD focuses on the desired behavior which then leads to tests
 - Associated with Cucumber
- ▶ Acceptance test-driven development (ATDD) focuses on the tests of the desired behavior
 - Associated with Fit/Fitnesse

cucumber

FitNesse

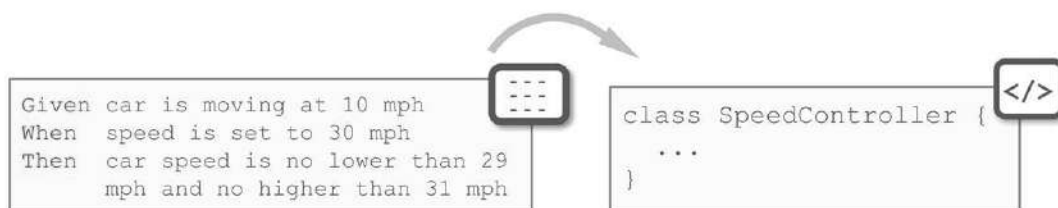
SCALED AGILE® © Scaled Agile, Inc.

10

Notes:

Think test-first


- ▶ No code goes in until the test goes on
 - Don't test code, code to the test
 - Tests represent a Story's details



SCALED AGILE® © Scaled Agile, Inc.

11

Notes:



Activity: Behavior-driven development (BDD) tests

Prepare
30 min

Share
5 min

Working in your teams, apply BDD to the Stories you have created.

- ▶ **Step 1:** Create some BDD tests from the criteria for the Stories previously written. Alternatively, you can come up with other criteria and develop examples or tests from those.
- ▶ **Step 2:** Discuss how the tests help detail the Stories.

SCALED AGILE® © Scaled Agile, Inc.

12

Notes:

Gherkin is a common syntax for Given/When/Then

Free text form

```
Given current speed is 40 mph
    and desired speed is 30 mph in 200 feet
    and the goal is mileage
When braking is applied
Then the deceleration rate is
    5 feet/sec^2 with
    jerk deceleration of 2 feet/sec^3
```

Table form

```
Given automobile has:
    | Current speed | Desired speed | Distance | Goal |
    | 40 mph       | 30 mph       | 200 feet | mileage |
When braking is applied
Then slowdown is
    | Deceleration rate | Jerk Deceleration |
    | 5 feet/sec^2      | 2 feet/sec^3      |
```



Which representation do you prefer?

Notes:

Other ways to represent tests

Gherkin - combined table form

```
Given current speed when braking applied then deceleration should be
| Current | Desired | Distance | Goal | Deceleration | Jerk
| speed   | speed   |          |      | rate          | Deceleration
| 40 mph  | 30 mph  | 200 feet | mileage | 5 ft/sec^2    | 2 ft/sec^3
| 50 mph  | 30 mph  | 200 feet | time   | 10 ft/sec^2   | 5 ft/sec^3
```

Fit/FitNesse form

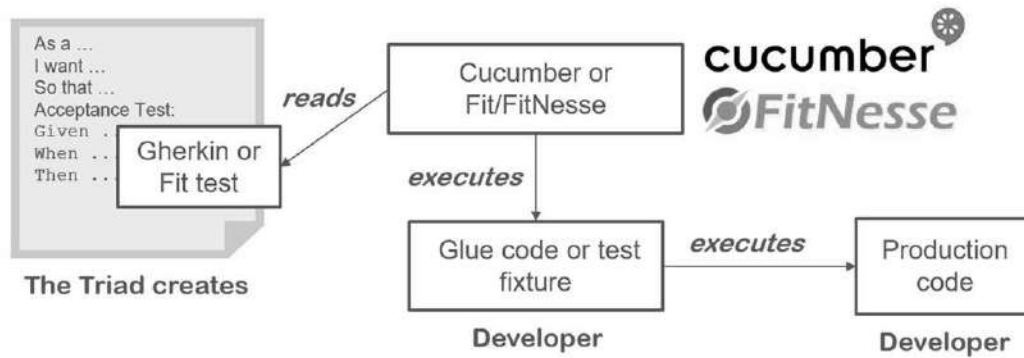
Current speed	Desired speed	Distance	Goal	Deceleration rate?	Jerk Deceleration?
40 mph	30 mph	200 feet	mileage	5 feet/sec^2	2 feet/sec^3
50 mph	30 mph	200 feet	time	10 feet/sec^2	5 feet/sec^3

Note: There are many BDD frameworks. These two are representative.

Notes:

Automate BDD tests for frequent and fast feedback

Decrease impedance between tests and code.



Notes:

7.2 Specify desired behavior for domain terms

SCALED AGILE © Scaled Agile, Inc.

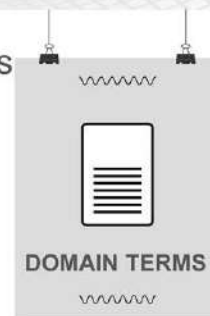
16

Notes:

Use domain terms to provide a common vocabulary

Domain terms are ubiquitous ways to talk about entities

- ▶ Avoid misunderstanding and miscommunication
- ▶ Similar concept to the *Data Dictionary*



What is 'frequency' for a sound? Is it the time between tones?
Or the Hz of the tone?

SCALED AGILE © Scaled Agile, Inc.

17

Notes:

Validate behavior for domain terms

- There are many ways to test a domain term's behavior
 - Min/max values, format (e.g., ###-##-###), range (0..50)




Example: *Define validation for 'Speed'*

	Speed		Valid?	
	0		Yes	
	149		Yes	
	150		No	
	-1		No	



Notes:

7.2 Specify desired behavior for domain terms



Activity: Behavior test for domain terms

Prepare
5 min

Share
5 min

- ▶ **Step 1:** Consider the domain (context) you have chosen
- ▶ **Step 2:** Write down the domain terms
(either from the *Autonomous Vehicle Parking* domain or from your own context)
- ▶ **Step 3:** Write a behavior test for a domain term

SCALED AGILE © Scaled Agile, Inc.

19

Notes:

7.3 Describe behavior for business rules and algorithms with tests

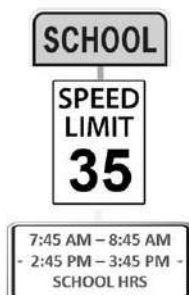
SCALED AGILE © Scaled Agile, Inc.

20

Notes:

Separate business tests and scenarios tests

- Scenario flows often have business rules
- Test variations of business rules separately from flow tests



Business rule

Obey speed limit variations in school zones

Business rule data

Start Time	Stop Time	Speed Limit
7:45 AM	8:45 AM	35 mph
2:45 PM	3:45 PM	35 mph

Tests for business rule


Time	Speed Limit	Notes
7:44 AM	45 mph	Speed limit for road
7:45 AM	35 mph	School zone applies

SCALED AGILE © Scaled Agile, Inc.

21

Notes:

7.3 Describe behavior for business rules and algorithms with tests



Discussion: Business rule variations

Discuss
3 min

Business rule data

Start Time	Stop Time	Speed Limit
7:45 AM	8:45 AM	35 mph
2:45 PM	3:45 pm	35 mph

How many test variations do we need to cover the business rule?

Tests for business rule

Time	Speed Limit	Notes
7:44 AM	45 mph	Speed limit for road
7:45 AM	35 mph	School zone applies
??	??	??
??	??	??

Which edge cases should we consider?
Do we slow down at 7:44 or 7:45?

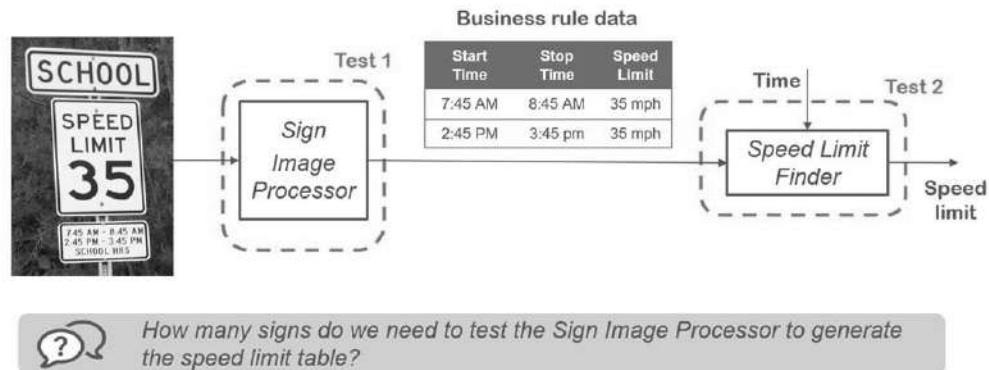
SCALED AGILE® © Scaled Agile, Inc.

22

Notes:

Create separate tests for algorithms and calculations

Example: Separate image recognition of signs from business rule data to determine speeds.



SCALED AGILE © Scaled Agile, Inc.

23

Notes:

Decompose large tests

User Story

As a driver, I want to enter a destination and get the next route instruction to that destination

Large test

Given automobile's location is 732 9th St
 And automobile's heading is south
 When the destination entered is 1802 W Main St
 Then next route instruction is turn left in .15 miles onto Main St

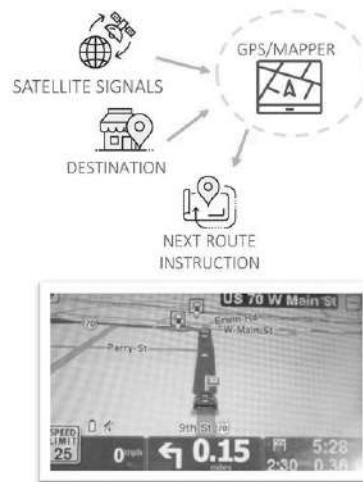


What are the issues with running this test?

Notes:

Issues with the large test

- ▶ The test is stateful
 - To run the test, the automobile must start at 732 9th Street
- ▶ The test would be slow and expensive
 - It should be run at least once to ensure the whole system works
- ▶ The test would work better as a stateless business rule/calculation/algorithm

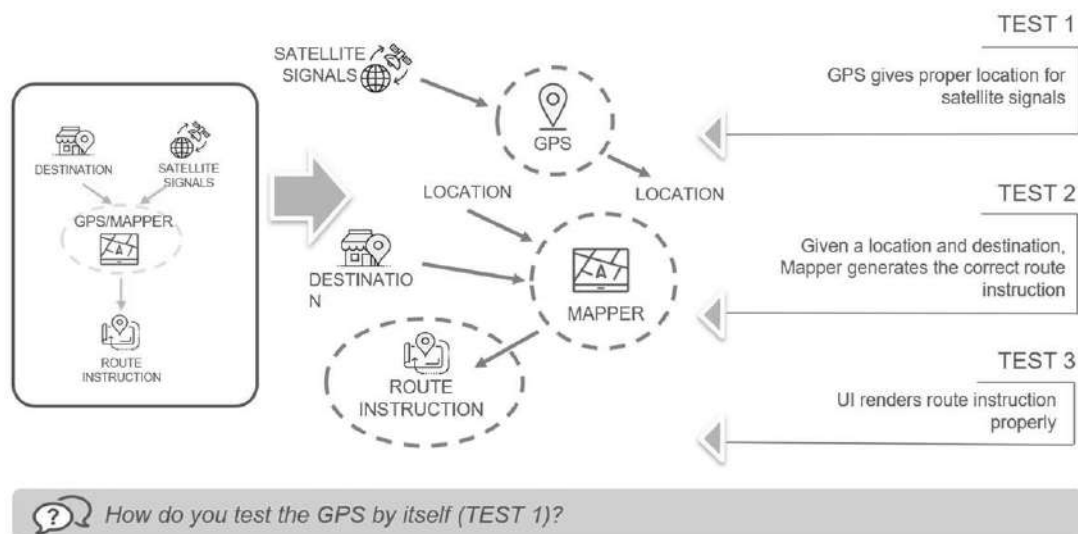


SCALED AGILE © Scaled Agile, Inc.

25

Notes:

Decompose large tests into multiple tests



Notes:

Mapper and display tests

Test 2: Stateless business rule/calculation test

Given automobile is at <Location> and heading is <Heading>

When going to <Destination>

Then route instruction is <Route Instruction>

Location	Heading	Destination	Route Instruction
732 9th Street	South	1802 Main Street	Left .15 miles Main Street
1802 Main Street	West	732 Ninth Street	Right .1 miles 9th Street

Test 3: Test for the display


Given route instruction is Left .15 miles Main Street

When displayed

Then display renders correct route instruction as shown



Notes:



Activity: Stateless tests

Prepare
5 min

Share
5 min

- ▶ **Step 1:** Consider the scenario test you have previously written.
- ▶ **Step 2:** Are there any business rules/calculations in this scenario? If so, create a test for one of the business rules.
- ▶ **Step 3:** Is your scenario test stateless? If not, how could you decompose it into at least one stateless test?

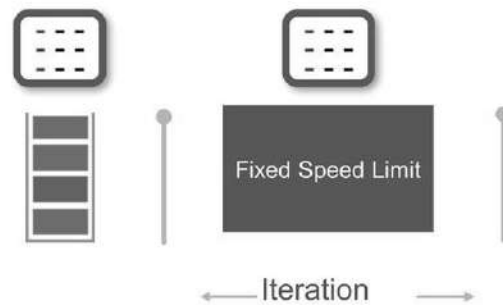
SCALED AGILE © Scaled Agile, Inc.

28

Notes:

Apply BDD continuously

- ▶ During Story refinement to improve estimates by detailed Stories
 - May add significant time to refinement session
- ▶ When implementing the Story to finalize specific Story details



Notes:

7.4 Test the user interface (UI)

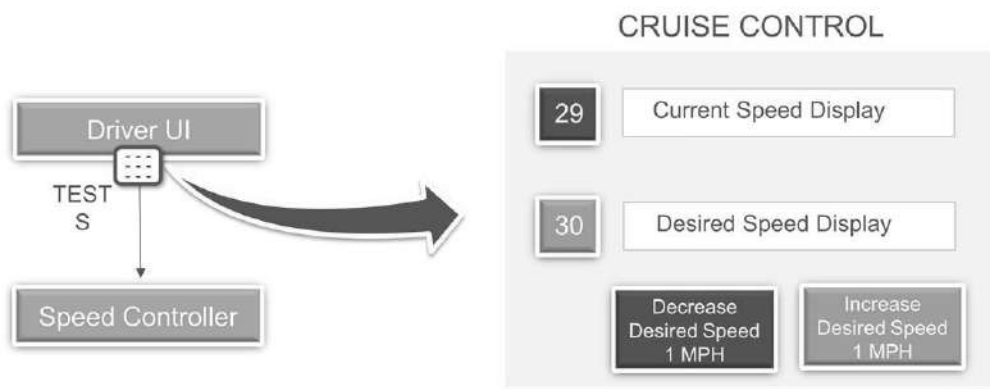
SCALED AGILE © Scaled Agile, Inc.

30

Notes:

Test user interface (UI) separately from the system

- ▶ Does the UI generate appropriate messages for inputs?
- ▶ Does UI render response correctly?



SCALED AGILE © Scaled Agile, Inc.

31

Notes:

Write separate BDD tests for UI and functionality

Tests UI and functionality

Given desired speed is 30
and current speed is 29
When increase is selected
Then desired speed becomes 31
and current speed should
become between 30 and 32
within 1 second



Test UI

Given desired speed is 30
When increase is selected
Then desired speed should
become 31



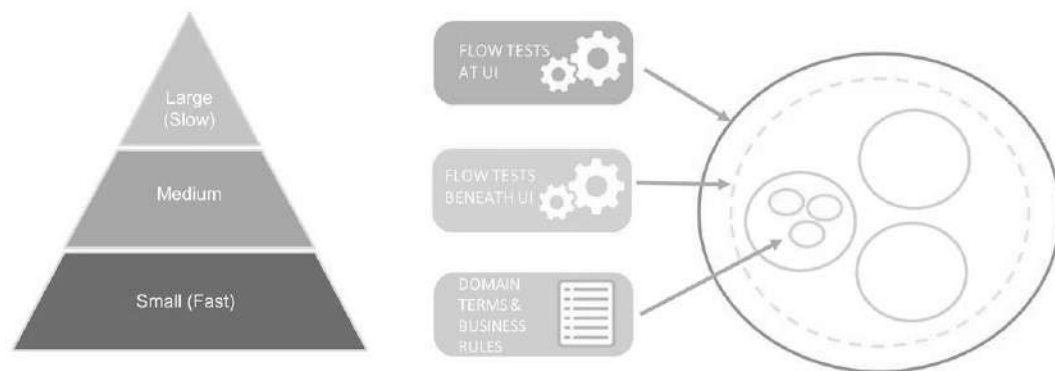
Test functionality

Given desired speed is 30
and current speed is 29
When increase is selected
Then current speed should
become between 30 and 32
within 1 second

Notes:

Do not test business rules/application logic through UI

- ▶ Testing UI is fragile: a change in UI often requires a change in tests
- ▶ Do not test business rules/calculations through UI
 - Test one variation in UI, and test remaining variations directly to business rule component




SCALED AGILE™ © Scaled Agile, Inc.

33


Notes:

7.4 Test the user interface (UI)



Discussion: User Interface (UI) tests

Discuss
3 min



How could you independently test the user interfaces (UI) for your Stories?

DISCUSS
3 min

SCALED AGILE® © Scaled Agile, Inc.

34

Notes:

7.5 Apply test doubles to BDD

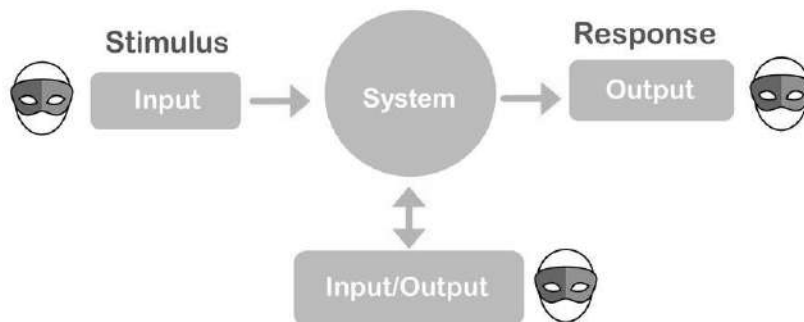
SCALED AGILE © Scaled Agile, Inc.

35

Notes:

Context diagram with test doubles

- ▶ Some test doubles can be used across an application
- ▶ Some can be used across Stories in a Feature



SCALED AGILE © Scaled Agile, Inc.

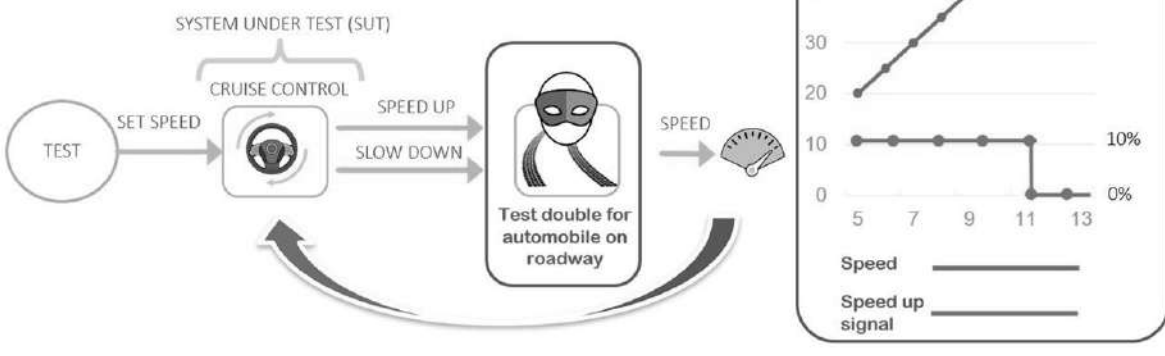
36

Notes:


Example: Test doubles for cruise control

► Roadway test double:

- Simulates roadway conditions (hills, curves, etc.)
- Responds to events based on how vehicle navigates the road (going uphill, downhill, tight turn, etc.)



Notes:



Discussion: Test double


Discuss
2 min

Share
3 min

► **Step 1:** At your table, discuss:

- What test doubles would be needed for a Story in your domain?
- How would you check if the test double represents reality?

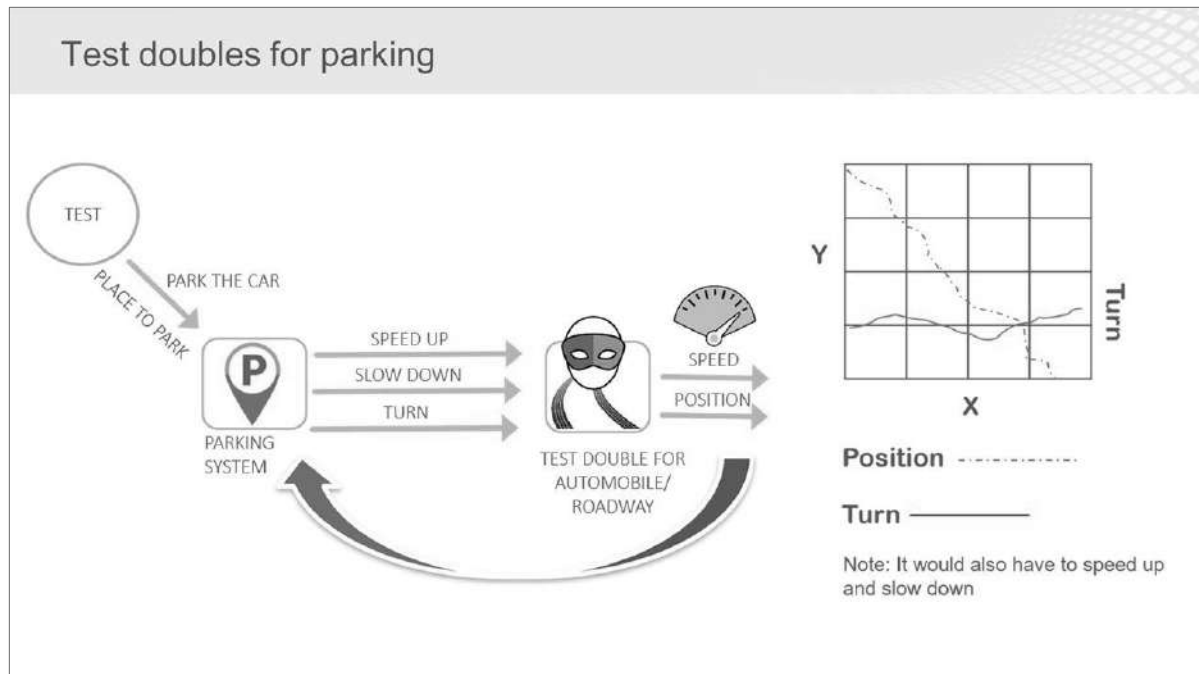
► **Step 2:** Be prepared to share with the class



SCALED AGILE® © Scaled Agile, Inc.

38

Notes:



Notes:



Notes:

7.6 Find existing tests impacted by new requirements

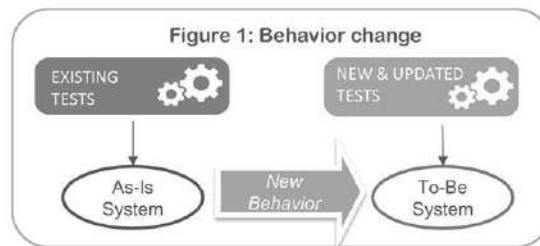


Discussion: Discovering impacted tests



- **Step 1:** Consider Figure 1 below
- **Step 2:** At your table, discuss how to find impacted tests on a behavior change

Note: As requirements change, you need to ensure that tests change



SCALED AGILE® © Scaled Agile, Inc.

41


Notes:

Tests help track requirement changes

- ▶ Need to find existing tests for requirements as they change
- ▶ Make tests findable/searchable:
 - Hierarchy: Group by Feature or functional category
 - Keywords: So tests are easier to find



Notes:



Activity: Build your Agile Software Engineer Plan

Share
2 min

- ▶ **Step 1:** Considering what you have learned, write down at least one improvement item in your Agile Software Engineer Plan
- ▶ **Step 2:** Share an item you wrote with the class

In this lesson, we:

- ▶ Explained the use of models
- ▶ Outlined static models
- ▶ Demonstrated CRC technique
- ▶ Outlined dynamic models
- ▶ Outlined state models

SCALED AGILE® © Scaled Agile, Inc.

43

Notes:

7.6 Find existing tests impacted by new requirements



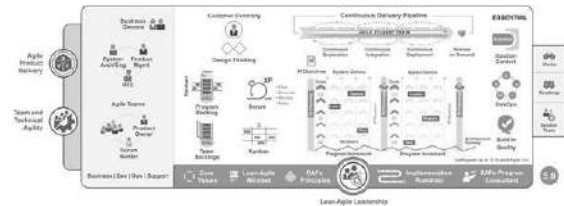
Discussion: BDD in SAFe

Discuss



► Discuss:

- Which role(s) in SAFe define BDD tests?
- In SAFe, when should BDD tests be defined?



SCALED AGILE® © Scaled Agile, Inc.

44

Notes:



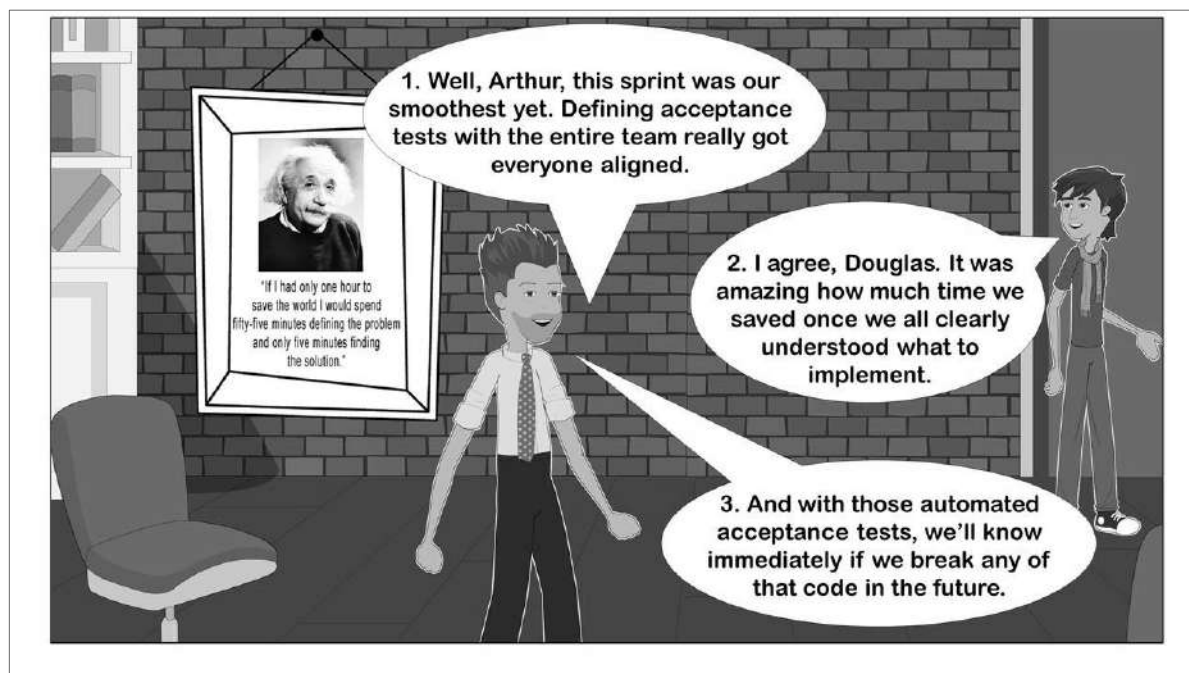
Agile Software Engineering Action Plan



- ▶ **Step 1:** Locate the Agile Software Engineer Action Plan at the end of your workbook
- ▶ **Step 2:** Considering what we have covered, write down at least one improvement item in your Agile Software Engineer Action Plan
- ▶ **Step 3:** Share the item you wrote with the class.



Notes:



Notes:

Lesson review

In this lesson you:

- ▶ Applying Behavior-Driven Development (BDD) for acceptance testing
- ▶ Specifying desired behavior for domain terms
- ▶ Writing tests for business rules and algorithms
- ▶ Testing the User Interface (UI)
- ▶ Applying test doubles to BDD
- ▶ Finding existing tests impacted by new requirement

Notes:

Lesson 8

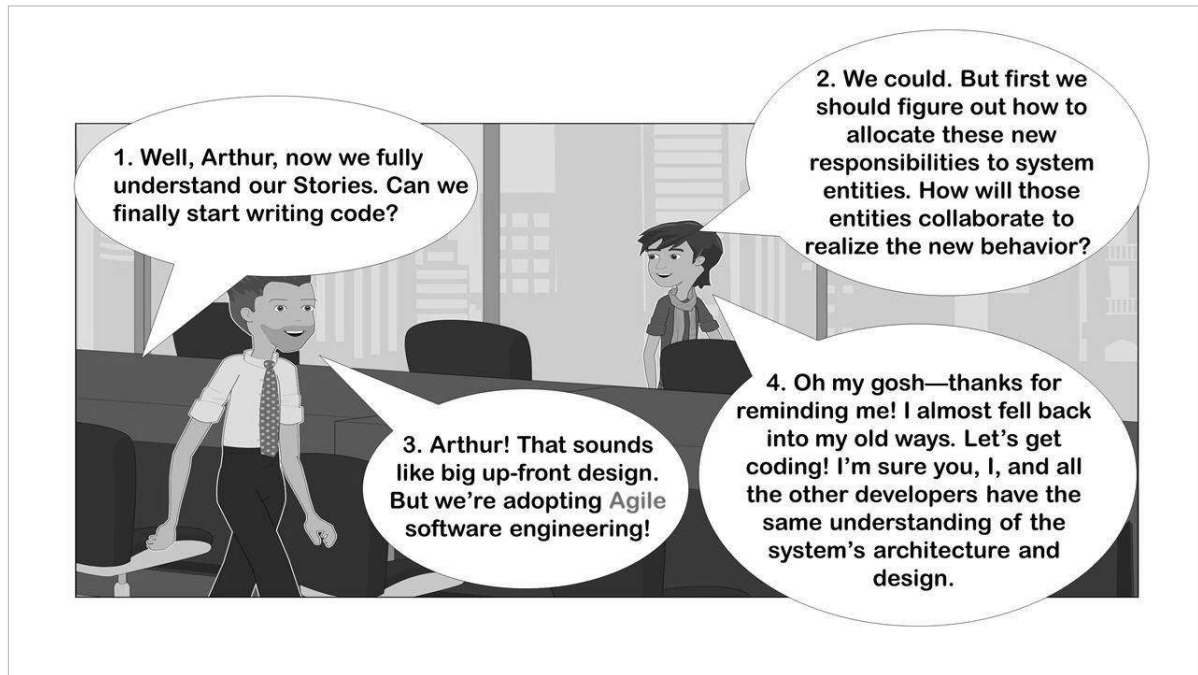
Communicating with Models

Learning Objectives:

- 8.1 Explain the use of models
- 8.2 Outline static models
- 8.3 Demonstrate class-responsibility-collaboration (CRC)
- 8.4 Outline dynamic models
- 8.5 Outline state models



SAFe® Authorized Course Attending this course gives students access to the SAFe® Agile Software Engineer exam and related preparation materials.



Notes:

Video: The importance of modeling

Duration
1 min

The importance of modeling
<https://vimeo.com/374274462/5adbd85290>

SCALED AGILE® © Scaled Agile, Inc.

222


Notes:

8.1 Explain the use of models

SCALED AGILE © Scaled Agile, Inc.

5

Notes:




Activity: What are examples of models?

Duration
3 min

- ▶ Some examples of models are...
- ▶ What did the model abstract from the real-world object, and what did it not capture? Why?
- ▶ What are some real-world examples of models that you have seen?

All models are wrong, but some are useful.

—George E. P. Box



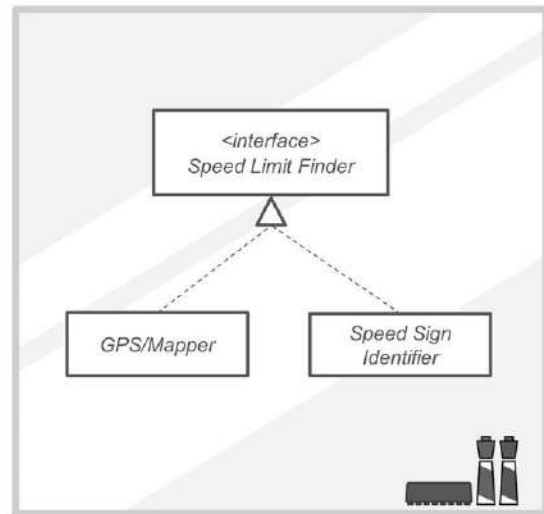
SCALED AGILE © Scaled Agile, Inc.

6

Notes:

Why model?

- ▶ Modeling helps create a common understanding
 - Aligns everyone on what the system does and will do
 - Takes a systems view by abstracting details not relevant to the current conversation
- ▶ Models provide a common vocabulary and overall design and architecture
 - Show domain terms, their relationship, and their communication



SCALED AGILE® © Scaled Agile, Inc.

7

Notes:

What is Agile modeling?

- ▶ Agile modeling is 'just enough modeling':
 - Code shows the details
 - Models can show connections between domain terms and implementation components



SCALED AGILE® © Scaled Agile, Inc.

8

Notes:

Models at various levels

► Story models

- Keep on a whiteboard as a current understanding of the small picture
- May change frequently
- Do not have to update once a story is done, and may be kept for reference

► Architecture models

- Maintain an understanding of the big picture
- Usually do not change frequently

Notes:

8.2 Outline static models

SCALED AGILE © Scaled Agile, Inc.

10

Notes:

Entity definitions

Entities can be:

- ▶ A single entity, such as a class
 - Typically these are small entity types
- ▶ Components or modules
 - Represent larger entity types
 - Are multiple entities or classes
 - Typically have one or more interfaces (components)
 - Are just a group of entities (modules)

Entity

Component



SCALED AGILE © Scaled Agile, Inc.

11

Notes:

Entity and entity type modeling

- ▶ An entity type
 - Is an automobile, for example
 - Can have attributes and behavior
- ▶ An entity is an instance of an entity type



Example:

Jane's car is an automobile entity with its own values for attributes

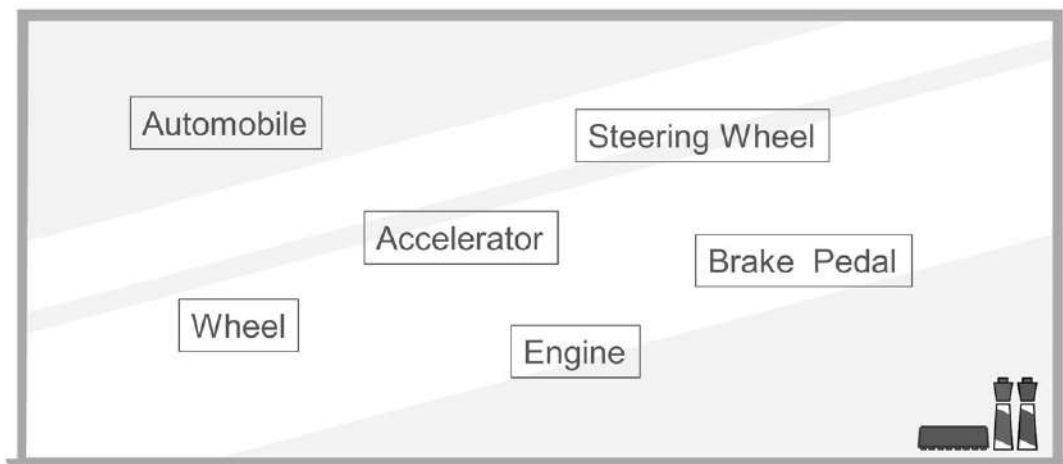


Example:

Bob's car is another automobile entity with its own values for attributes

Notes:

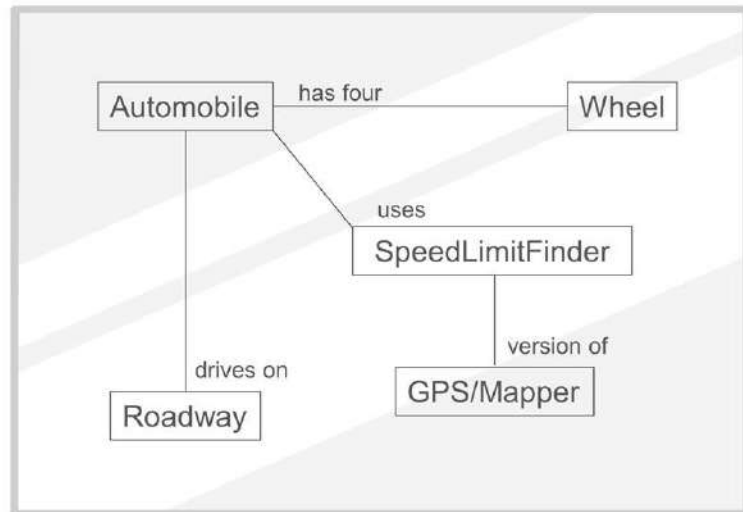
Entities may represent domain terms



Notes:

Relationships between entities

- Relationships can use words
- Alternatively, you can use common symbols to represent relationships



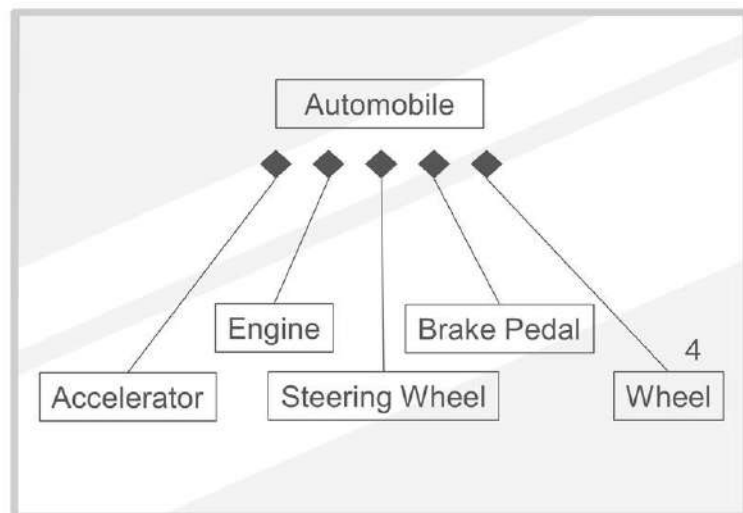
SCALED AGILE® © Scaled Agile, Inc.

14

Notes:

Composition of entities

- Automobile composed of these entities
- If number not shown, count is 1
- Otherwise, number shown can be number or * (any number) or range



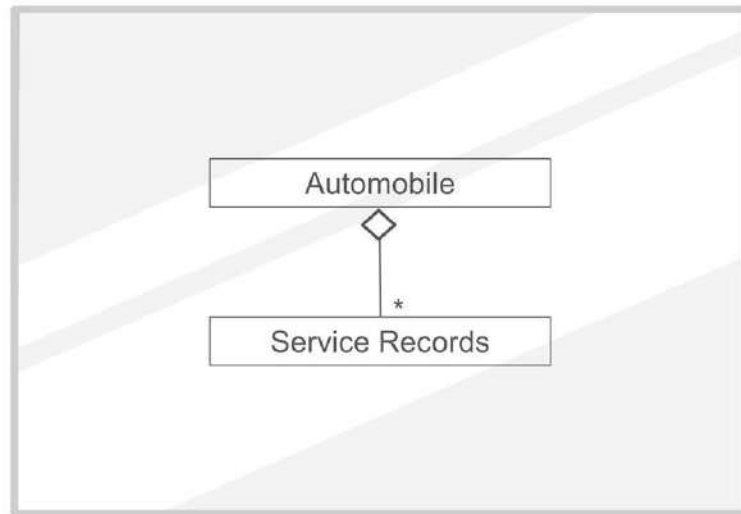
SCALED AGILE® © Scaled Agile, Inc.

15

Notes:

Aggregation

- Aggregation is like composition, but contained entities typically exist independently.



SCALED AGILE[®] © Scaled Agile, Inc.

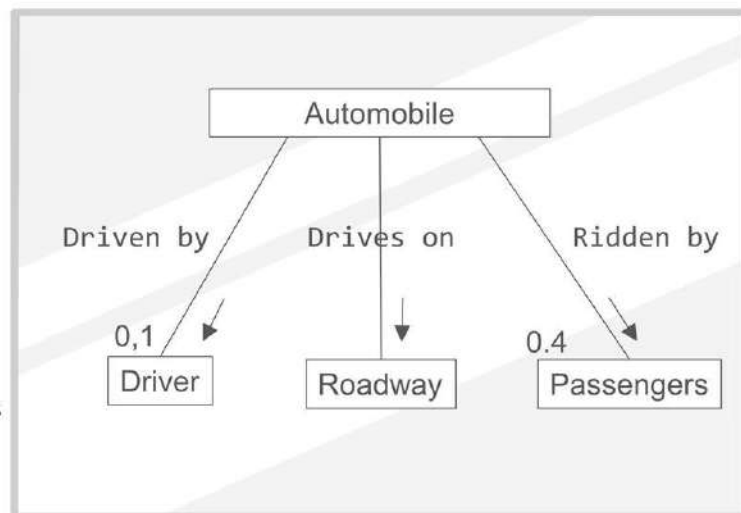
16

Notes:

Association of entities

- Association shows relationship between two entities
- Can describe in either direction

Driven by → or ← Drives



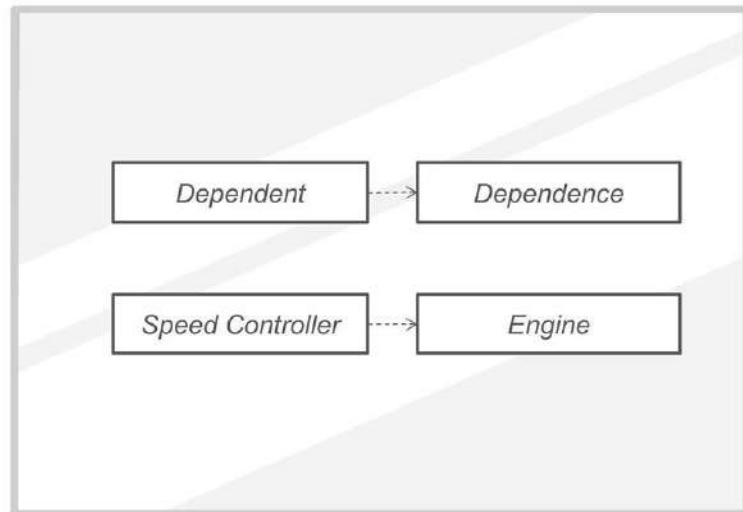
SCALED AGILE[®] © Scaled Agile, Inc.

17

Notes:

Dependency

- ▶ One entity uses another entity to help perform its responsibility
- ▶ Static diagrams can reflect dependencies, but tend to get messy
- ▶ Dynamic diagrams (shown later) also demonstrate dependencies
- ▶ **Example:** Speed controller depends on engine to accelerate car



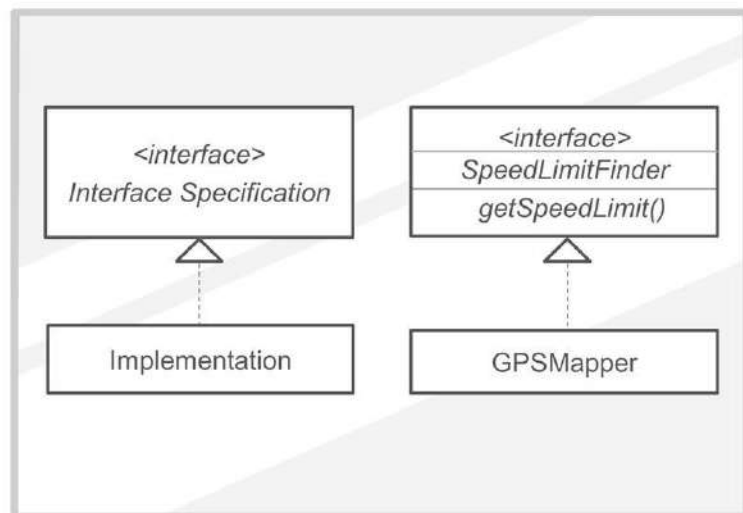
SCALED AGILE[®] © Scaled Agile, Inc.

18

Notes:

Implements

An implementation implements all the responsibilities (public methods) in an interface.



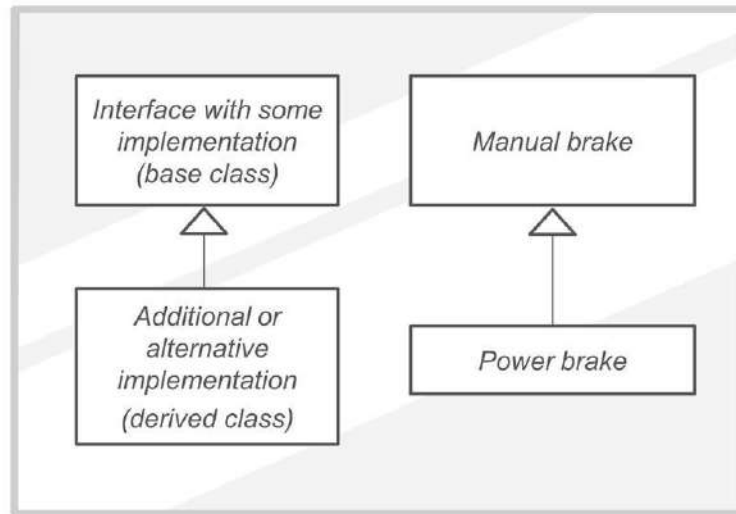
SCALED AGILE[®] © Scaled Agile, Inc.

19

Notes:

Inheritance


Inheritance is a form of implementation where the implementation can be shared.



SCALED AGILESM © Scaled Agile, Inc.

20

Notes:

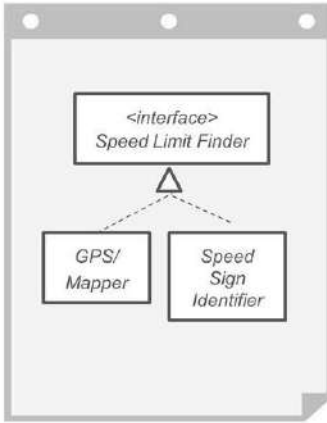


Activity: Static constituents modeling

Create
10 min

Review
4 min

- ▶ Step 1: In your teams, construct a diagram of the entities from some of your Stories using a flip chart sheet
 - Particularly the Story for which you created acceptance tests
- ▶ Step 2: Pair with someone from another team and review each other's diagrams



SCALED AGILE® © Scaled Agile, Inc.

21

Notes:

8.3 Demonstrate class-responsibility-collaboration (CRC)

SCALED AGILE © Scaled Agile, Inc.

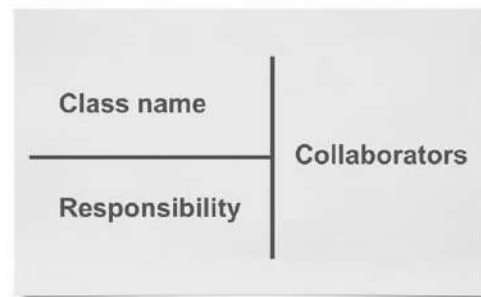
22

Notes:

Class-responsibility-collaboration (CRC) cards

► CRC cards show:

- **Class name** - Can also be component or other entity
- **Responsibility** - What the entity should be responsible for
- **Collaboration** - What other entities are needed to fulfil responsibilities



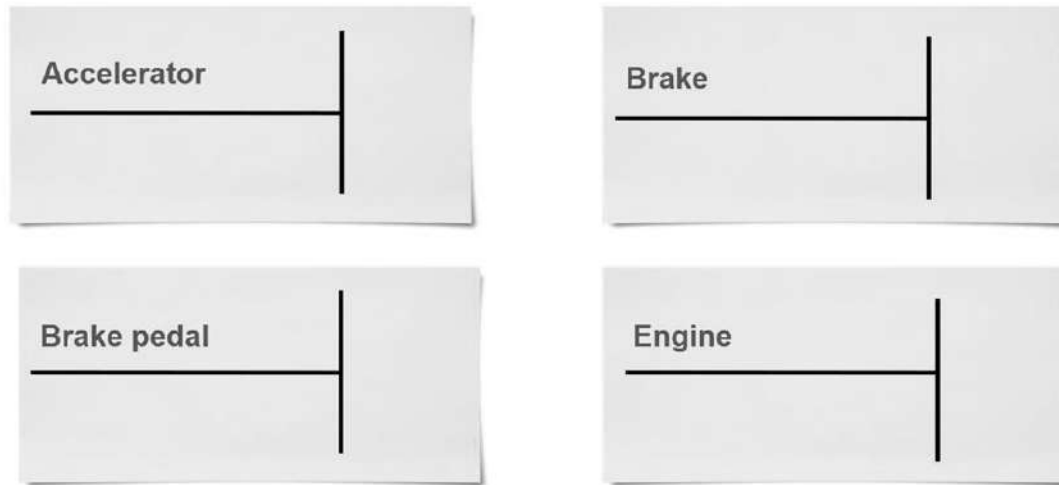
SCALED AGILE © Scaled Agile, Inc.

23

Notes:

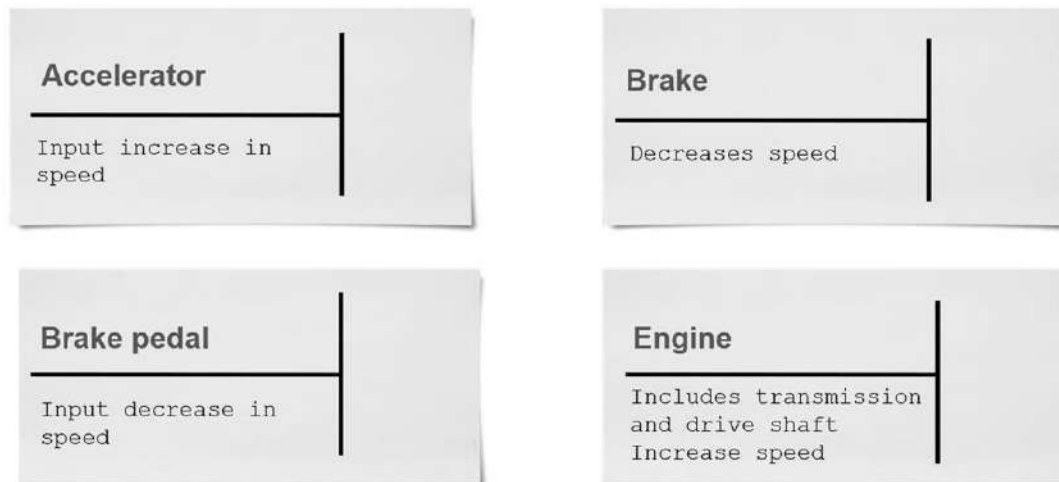
8.3 Demonstrate class-responsibility-collaboration (CRC)

Start with the entities



Notes:

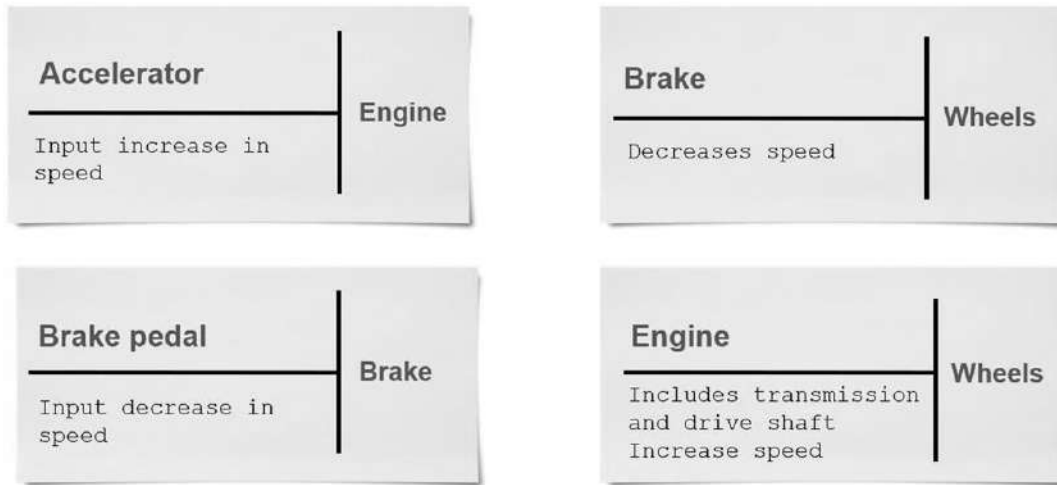
Assign responsibilities



Notes:

8.3 Demonstrate class-responsibility-collaboration (CRC)

Assign collaborators



Notes:

Carefully consider allocation of responsibilities

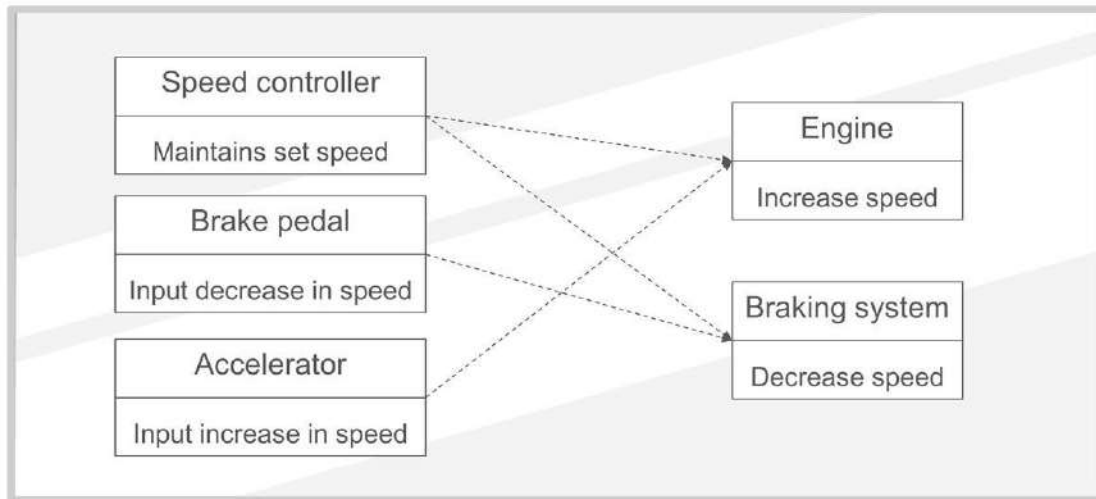
What should be responsible for maintaining speed: Brake, Engine, or another entity?



Notes:

8.3 Demonstrate class-responsibility-collaboration (CRC)

Collaborations appear as relationships in static diagrams




SCALED AGILESM © Scaled Agile, Inc.

28

Notes:

8.3 Demonstrate class-responsibility-collaboration (CRC)



Activity: Assigning responsibilities

Prepare
13 min

Share
2 min

- ▶ **Step 1:** Using entities created in the previous exercise, create CRC cards for them
- ▶ **Step 2:** Assign responsibilities and identify collaborators for each of them
 - You may discover additional entities that would be helpful
- ▶ **Step 3:** Discuss if there are alternative ways of assigning responsibilities (Rule of Three)

Class name	Collaborators
Responsibility	

SCALED AGILE® © Scaled Agile, Inc.

29

Notes:

8.4 Outline dynamic models

SCALED AGILE® © Scaled Agile, Inc.

30

Notes:

Purpose of dynamic models

Dynamic models show:

- ▶ Sequence of interactions between entities to fulfill a scenario
- ▶ How responsibilities are delegated to collaborators (from CRC)

Create dynamic models if interactions are numerous or complex.

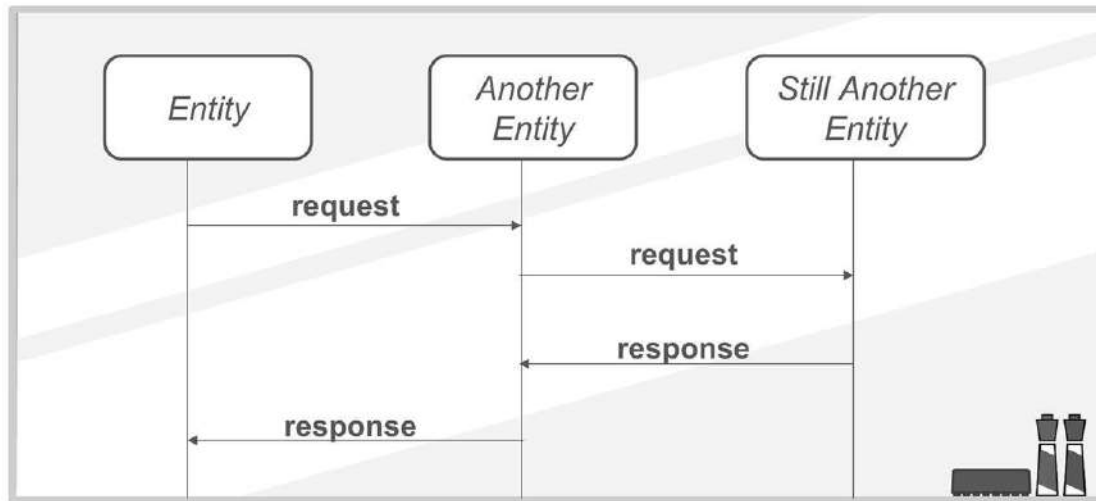
Note: The format of dynamic models shown here is minimalistic.

SCALED AGILE® © Scaled Agile, Inc.

31

Notes:

Demonstrate sequence diagram for scenario

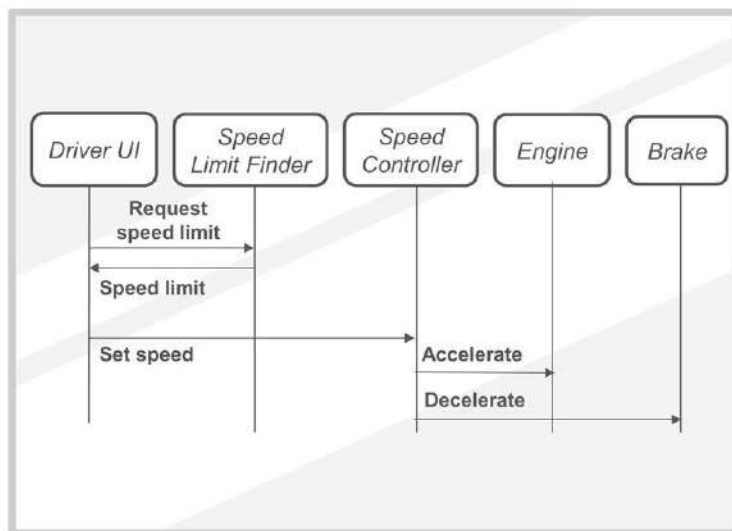


SCALED AGILE[®] © Scaled Agile, Inc.

32

Notes:

Sequence diagrams show collaboration to realize a scenario



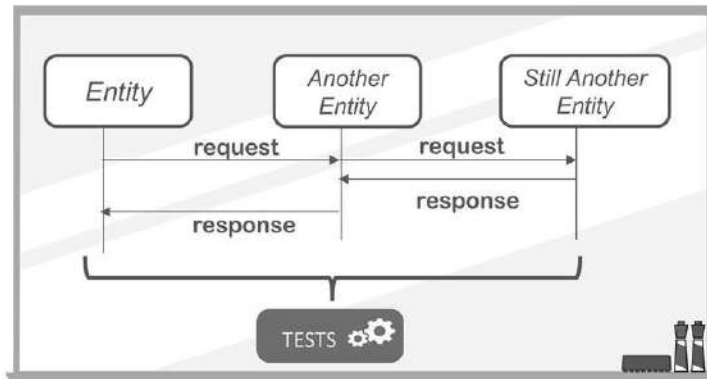
SCALED AGILE[®] © Scaled Agile, Inc.

33

Notes:

Testing scenarios

- ▶ Every sequence diagram should have a test
- ▶ But not every test needs a sequence diagram



SCALED AGILE® © Scaled Agile, Inc.

34

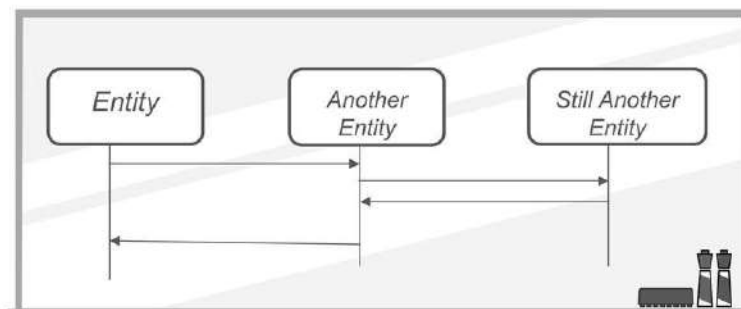
Notes:



Activity: Sequence of operations



- ▶ **Step 1:** You have already identified the collaborations in the previous activity. For that same story, draw a sequence diagram for the entities.
- ▶ **Step 2:** Discuss if there are alternative ways to sequence the operations.



SCALED AGILE® © Scaled Agile, Inc.

35

Notes:

8.5 Outline state models

SCALED AGILE © Scaled Agile, Inc.

36

Notes:

Example of state-dependent behavior

Entities (otherwise known as objects) often:

- ▶ Have states and go through transitions between states



Example: *States of automobile*

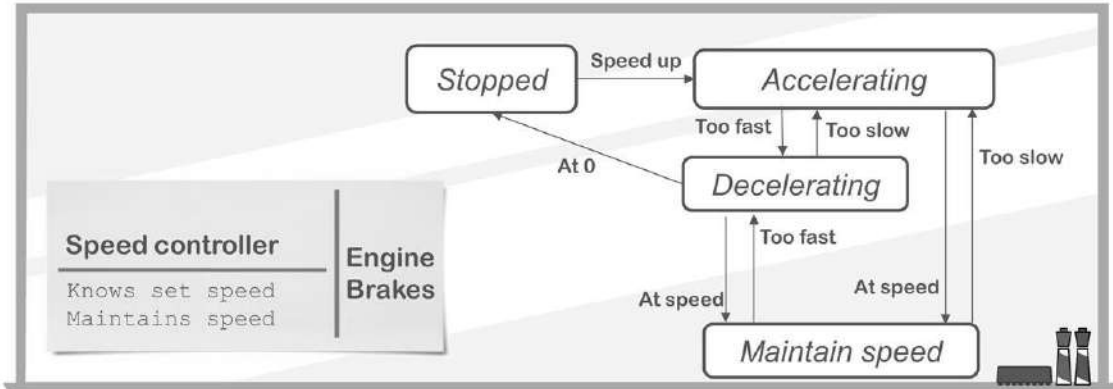
- *Stopped*
- *At speed*
- *Accelerating*
- *Decelerating*

- ▶ May behave differently based on what state they are in
 - Evaluate their condition
 - Then act or transition based on events

Notes:

Model state-dependent behavior with state models

- Speed controller responds differently depending on its state



SCALED AGILE® © Scaled Agile, Inc.

38

Notes:

State transition table shows transitions between states

- Table can be used in place of the diagram
- Shows non-applicable state/transition pairs (N/A)

State/Event	Speed up	Too Slow	Too Fast	At Speed	At 0
Stopped	Accelerating	N/A	N/A	N/A	N/A
Accelerating	N/A	N/A	Decelerating	Maintain Speed	N/A
Decelerating	N/A	Accelerating	N/A	Maintain Speed	Stopped
Maintain Speed	N/A	Accelerating	Decelerating	N/A	N/A



Are we missing any transitions?

SCALED AGILE® © Scaled Agile, Inc.

39


Notes:

An event causes a response within the receiving entity


- ▶ Responding to an event may cause an entity to:
 - Change its state
 - Perform an operation
 - Ask another entity to perform an operation
- ▶ Every state/event transition should have a test for:
 - The correct state transition
 - The correct response


Given state is Maintain Speed
 When event Too Fast occurs
 Then state becomes Decelerating
 ask Brake to deaccelerate

Notes:



Activity: State models

Prepare


Share


▶ **Step 1:** Review the Stories (form your own domain or *Autonomous Vehicle Parking*)

▶ **Step 2:** Create either a state diagram or a state transition table for any states in your Stories

! Alternative: If there is no appropriate example from your domain, complete the table for a parking spot instead


State/Event	Complete Car Parking
Empty	Filled			
Filled				
...				

Notes:

Modeling for shared understanding

When and Where to Model	Why
Architectural level	To understand and communicate the big picture
Feature level	To understand and communicate higher-level concepts
Story level	To understand the specifics of a Story
Just-in-time	To communicate and create shared understanding

Notes:



Activity: Build your Agile Software Engineer Plan

Share
2 min

- ▶ **Step 1:** Considering what you have learned, write down at least one improvement item in your Agile Software Engineer Plan
- ▶ **Step 2:** Share an item you wrote with the class

In this lesson, we:

- ▶ Explained the use of models
- ▶ Outlined static models
- ▶ Demonstrated CRC technique
- ▶ Outlined dynamic models
- ▶ Outlined state models

SCALED AGILE® © Scaled Agile, Inc.

43

Notes:

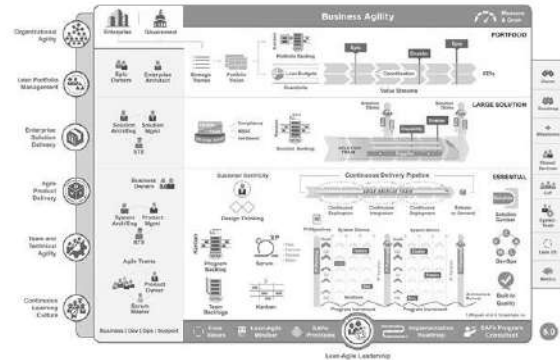


Discussion: Modeling stories in SAFe

Discuss



- Discuss where in SAFe you would apply modeling.
 - Which roles would collaborate using models?
 - What would they want to understand or communicate?



SCALED AGILE® © Scaled Agile, Inc.

44

Notes:



Agile Software Engineering Action Plan



- ▶ **Step 1:** Locate the Agile Software Engineer Action Plan at the end of your workbook
- ▶ **Step 2:** Considering what we have covered, write down at least one improvement item in your Agile Software Engineer Action Plan
- ▶ **Step 3:** Share the item you wrote with the class.



Notes:



Notes:

Lesson review

In this lesson you:

- ▶ Explaining the use of models
- ▶ Outlining Static models
- ▶ Demonstrating CRC technique
- ▶ Outlining Dynamic models
- ▶ Outlining State models

Notes:

Lesson 9

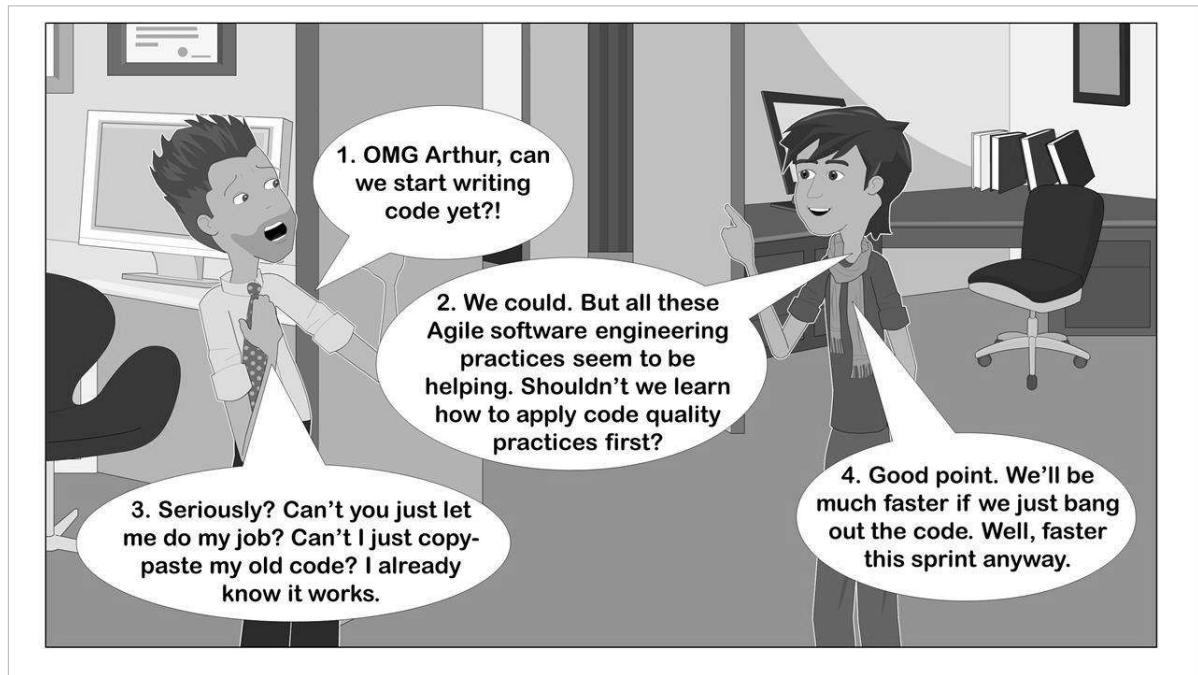
Building Systems with Code Quality

Learning Objectives:

- 9.1 Identify code qualities
- 9.2 Describe cohesion and coupling
- 9.3 Describe other code qualities
- 9.4 Explain the benefits of collective ownership



SAFe® Authorized Course Attending this course gives students access to the SAFe® Agile Software Engineer exam and related preparation materials.



Notes:



Notes:

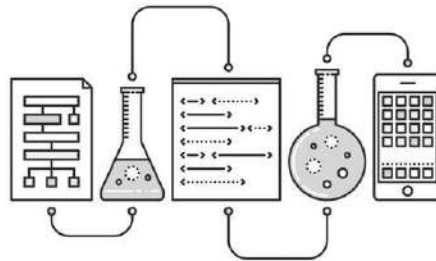


Discussion: Code quality

Discuss



- **Step 1:** What do you look for in a code review (either formal or in pairing)?
- **Step 2:** What issues you have experienced with maintaining code?



SCALED AGILE® © Scaled Agile, Inc.

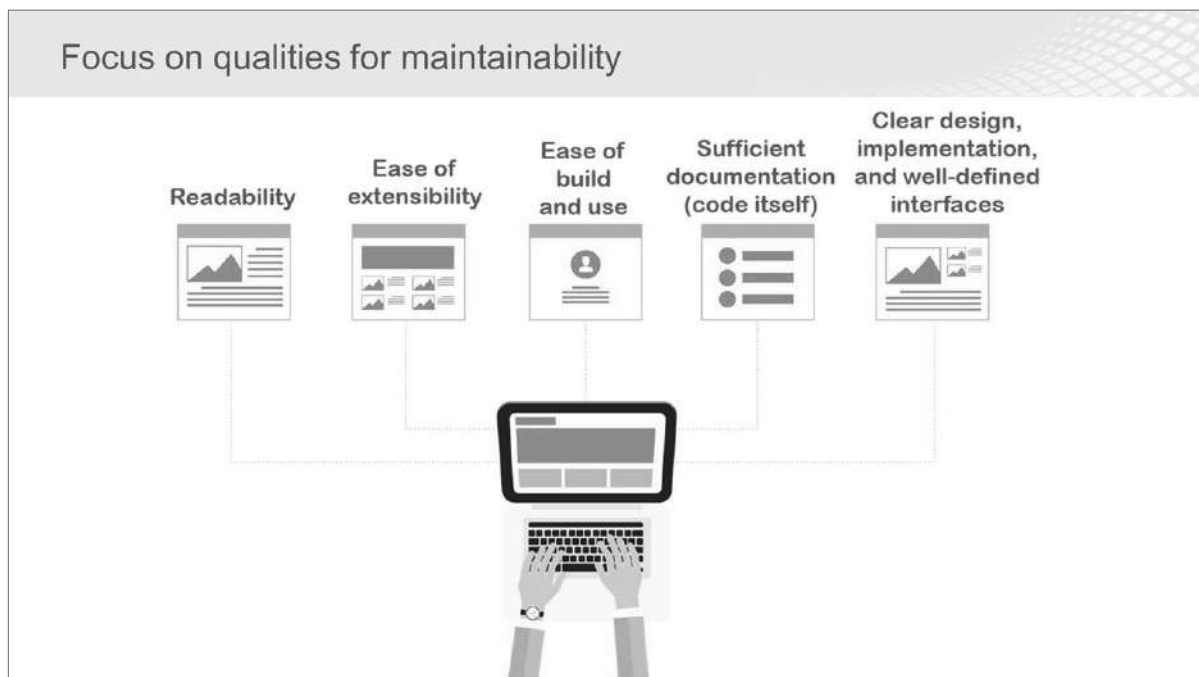
5

Notes:

9.1 Identify code qualities



Notes:



Notes:

What is the basis for application qualities?



Notes:

9.2 Describe cohesion and coupling

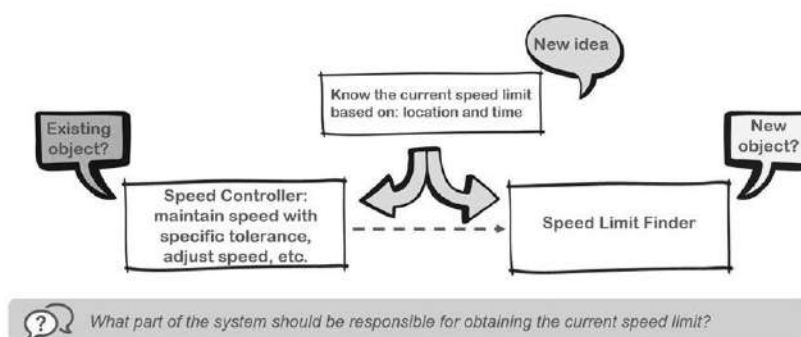
SCALED AGILE © Scaled Agile, Inc.

9

Notes:

Understanding cohesion: Assigning responsibilities

Given a new responsibility, where is the **best** space to assign it?
The current object, another object, or a new object?



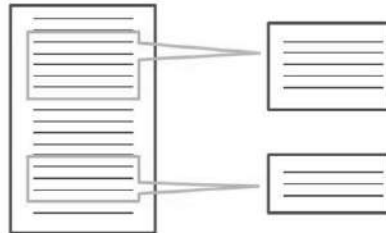
SCALED AGILE © Scaled Agile, Inc.

10

Notes:

Splitters and lumpers

When in doubt, split responsibilities into separate entities. Then lump if there's too much splitting.



Guideline: Splitters can be lumped more easily than lumpers can be split

SCALED AGILE® © Scaled Agile, Inc.

11

Notes:

Cohesion may have alternatives

Concept/Language	English	German
Welcome	Hello	Guten tag
Departure	Goodbye	Auf weidersehen

Concept/Language	English	German
Welcome	Hello	Guten tag
Departure	Goodbye	Auf weidersehen

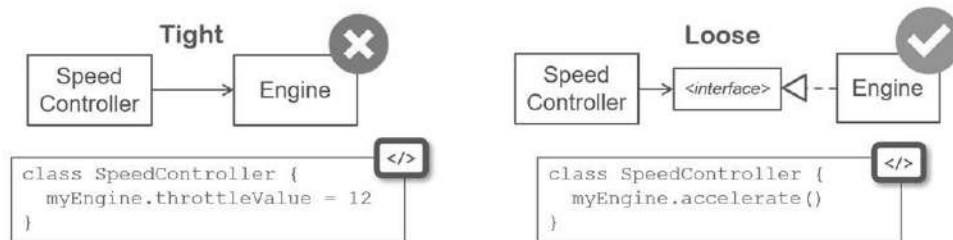


Which are more cohesive, concepts or languages?

Notes:

Types of coupling

- ▶ None: No knowledge of other entity
- ▶ Tight: Aware of other entity's implementation
 - Dependent on internal details
- ▶ Loose: Entities communicate through well-defined interface
 - Dependent on methods and arguments



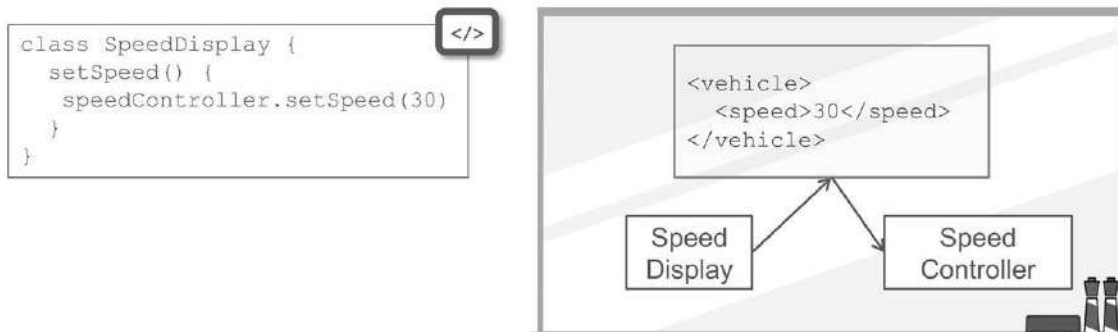
SCALED AGILE® © Scaled Agile, Inc.

13

Notes:

Coupling on methods or data

- ▶ Method interfaces coupled on technology (e.g. Java, C#, etc.)
- ▶ Data interfaces more loosely coupled (not a method invocation)



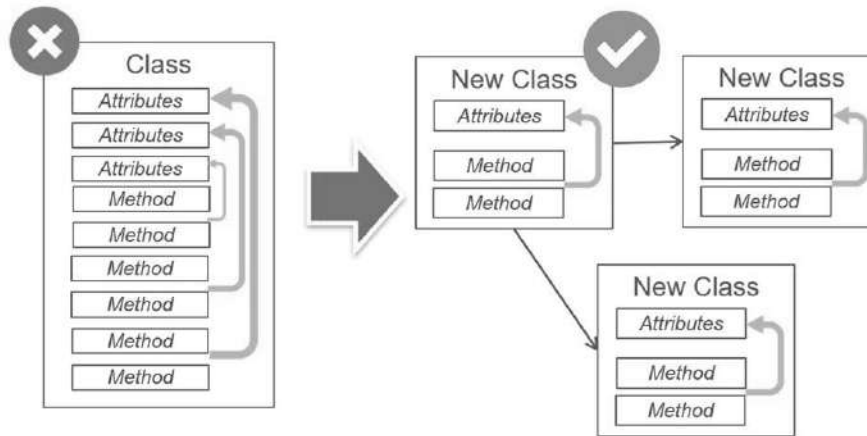
SCALED AGILE® © Scaled Agile, Inc.

14

Notes:

Reduce coupling within a class

Large classes with lots of attributes typically have lots of coupling.



SCALED AGILE[®] © Scaled Agile, Inc.

15

Notes:

Design guideline on structure

Favor delegation over inheritance.
—Design Patterns



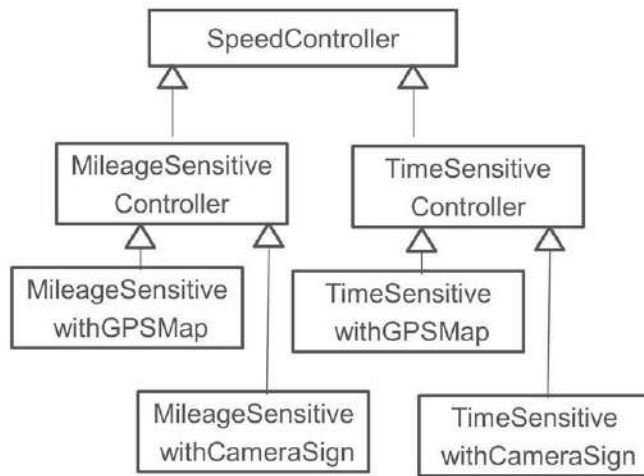
- ▶ Inheritance is tight coupling
- ▶ Avoid premature hierarchization
 - Create hierarchies when they emerge

SCALED AGILE[®] © Scaled Agile, Inc.

16

Notes:

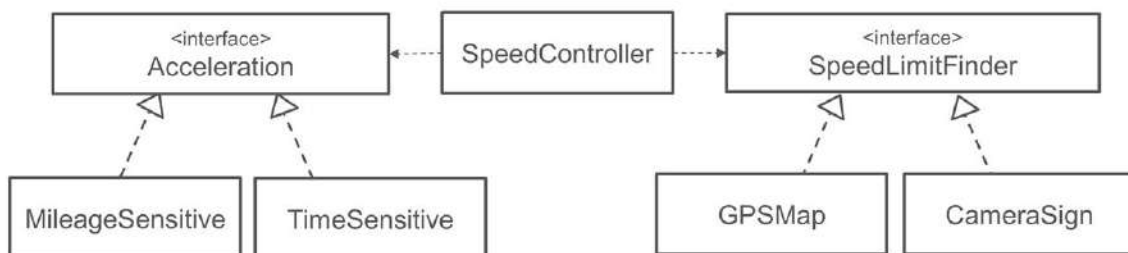
Inheritance



What if there were five acceleration types and five speed limit finders?

Notes:

Delegation

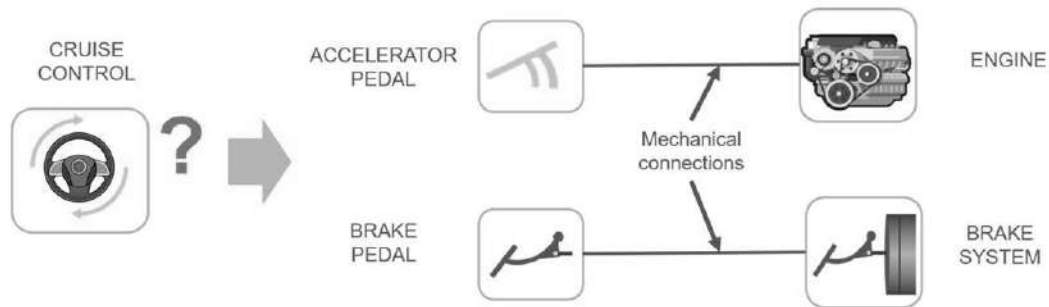


DEFINITION

Interface: Defines the behavior that any implementation must implement. It is a contract between an implementation and clients.

Notes:

Implementing new functionality in tightly coupled systems

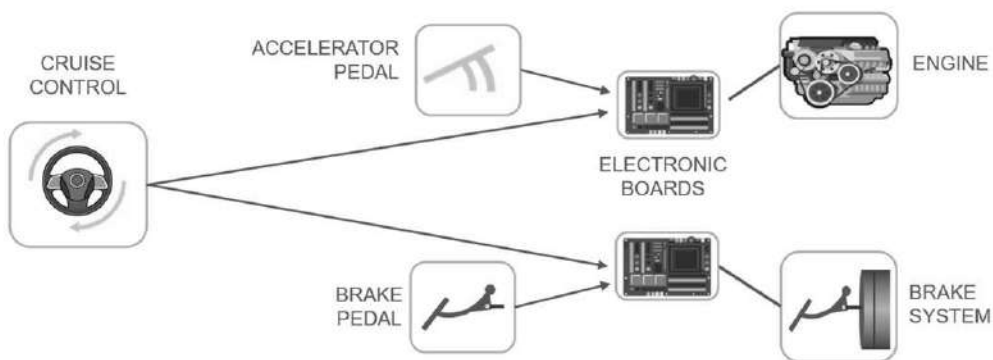


How could we add cruise control to the above system?

Notes:

Decoupled systems are easier to extend

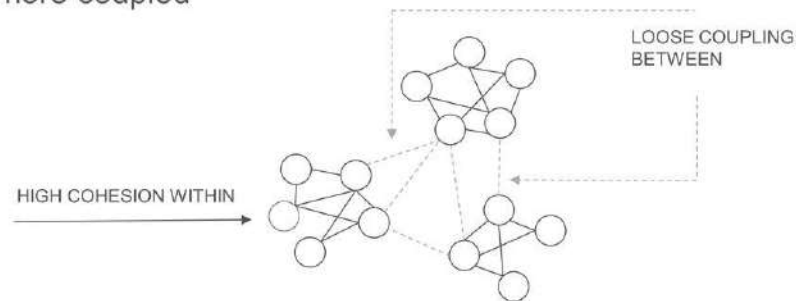
*We can solve any problem by introducing an extra level of indirection (abstraction)
—Fundamental Theorem of Software Engineering*



Notes:

Create cohesive entities with loose coupling


- Responsibilities are closely related to cohesion and coupling
 - They guide how we decompose components, classes, methods, etc.
 - Entities within a group that serves a common responsibility may be more coupled




SCALED AGILE® © Scaled Agile, Inc.


21

Notes:



Activity: Have you seen unwanted coupling?





Unwanted coupling...

Have you seen unwanted coupling?

What problems did it cause and how did you mitigate those problems?

SCALED AGILE® © Scaled Agile, Inc.

22

Notes:

9.3 Describe other code qualities

SCALED AGILE © Scaled Agile, Inc.

23

Notes:

Non-redundancy

- ▶ Redundancy: Two or more code modules having the same function or managing the same data
- ▶ Non-redundancy: Lacking redundancy—also called, 'don't repeat yourself' and 'once and only once'
- ▶ If change is made to function or data, it has to be made in two or more places
- ▶ The easiest code to debug is that which is not created



Avoid COPY AND PASTE

In code, tests, and requirements, copying and pasting is the quickest way to introduce redundancy

SCALED AGILE © Scaled Agile, Inc.

24

Notes:

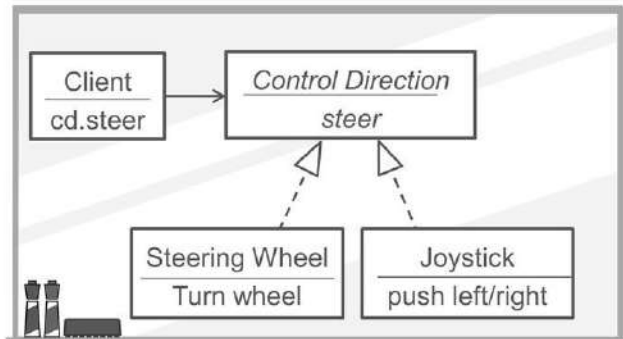
Abstraction and encapsulation

► Abstraction

- Uses domain terms that are not dependent on implementation

► Encapsulation

- Hides implementation
- Helps create abstraction and eliminates tight coupling



Question: Think about a phone. Do you dial a number on a phone? Is there a different name that is more abstract?

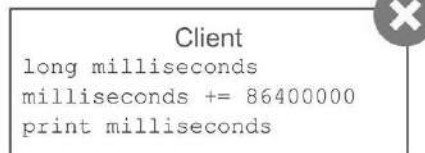
SCALED AGILE® © Scaled Agile, Inc.

25

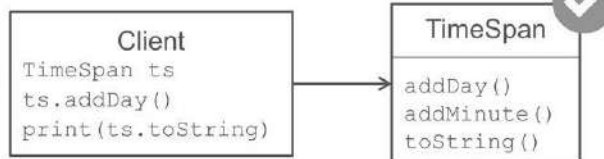
Notes:

Encapsulate data and behavior with abstract data types

Non-abstract type (primitive)



Abstract data type





What are some advantages ADTs provide developers over primitive types?

Notes:

Abstract data types (ADTs)

- ▶ Ensure quality, provide readability
- ▶ Detect improper use of data types
- ▶ Help prevent 'Poka Yoke' (inadvertent mistakes)

Primitives	Abstract data types (ADT)
 <pre>double computeSpeed (double distance, double time) double time double distance double current current = computeSpeed(distance, time) # What else can you do? distance * distance / speed * time speed = computeSpeed(time, distance)</pre>	 <pre>Speed computeSpeed (Length distance, TimeSpan time) TimeSpan time Length distance Speed current current = computeSpeed(distance, time) # What else can you NOT do? distance * distance / speed * time speed = computeSpeed(time, distance)</pre>

SCALED AGILE[®] © Scaled Agile, Inc.

27

Notes:

Use ADTs to support validation

- ▶ ADTs contains logic instead of every client
 - Ensures validity and consistency for all states and behaviors

Primitives	Abstract data types (ADT)
 <pre>int speed = -1 setSpeed (speed)</pre>	 <pre>Speed sp = new Speed(-1) setSpeed(sp)</pre>



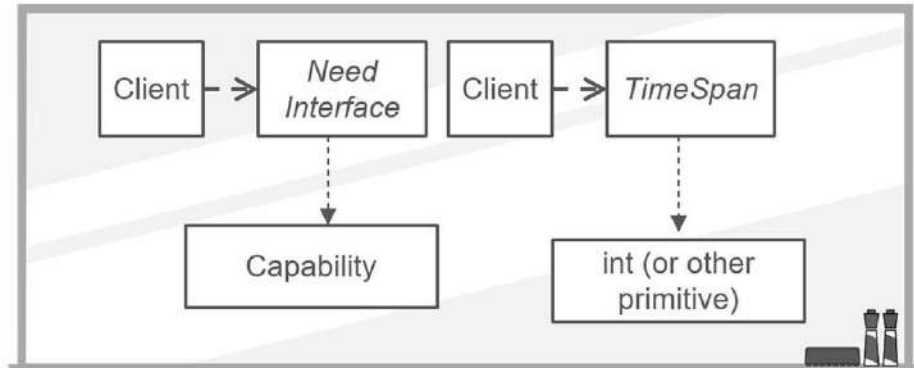
In each case, whose responsibility is it to ensure that speed is valid?

SCALED AGILE[®] © Scaled Agile, Inc.

28

Notes:

ADTs apply needs/capabilities separation to primitives



Notes:

Extreme abstraction

- ▶ When you are abstract, be abstract all the way
 - All domain terms should have an ADT
 - Don't use primitives in interfaces (e.g., method signatures)
 - Strings are primitives (if formatting or validation are important)
- ▶ It is easier to de-abstract for performance than it is to add abstraction
 - Performance overhead for abstraction varies by language and compiler optimization techniques



Example: C++ can have no overhead for abstraction

SCALED AGILE[®] © Scaled Agile, Inc.

30

Notes:



Discussion: Abstraction eliminates redundancy



- Look at the code example below.
 - Who is responsible for round-off in each case? (e.g. amount 100.01 interestRate .01)
 - Would there be duplication of the round-off algorithm in other methods?

Primitives

```
double amount
double interestRate
double interest
interest = amount * interestRate // What about round-off?
```



ADTs

```
Dollar amount
Percentage interestRate
Dollar interest = amount.multiplyBy(interestRate)
```



Notes:

Use domain names to improve quality

A rose by any other name is not a rose 

Each concept in a system should:

Have a clearly defined name

Code names should:

Match domain names
(Domain-driven design)

Less impedance between
domain and Solution

Extreme
readability



Guidance: Don't change what a term means

- Create new terms rather than applying new meanings to current terms
- Create different names for different things

Notes:

Improve readability with quality comments

Longer names for interfaces between entities

Short names where scope is small (within method)

Comment for why, not what (for intent not apparent from the code)

Comments could include:


- One-sentence description (for the intent of the class)
- List of responsibilities (from the CRC cards)




Notes:

Ensure names support readability

- ▶ Named constants remove the problem of magic numbers
 - Make code more readable, eliminate ambiguity, and avoid duplication



```
if (speed > 120)
...
if (speed <= 120)
```



```
constant MAXIMUM_SPEED = 120
if (speed > MAXIMUM_SPEED)
...
if (speed <= MAXIMUM_SPEED)
```



Guideline: No constant should appear in executable code

Notes:

Code readability improves quality

- ▶ Communicate with your code
 - Code is written once, read often
- ▶ Spell-check code, so it's readable (and searchable)



```
GetSomeField()
SetSomeFeild()
```



Do you spell-check your code (not just the comments)?

Notes:



Video: Example of assertive vs inquisitive code

Duration
1 min



<https://vimeo.com/374275271/88742cbde4>

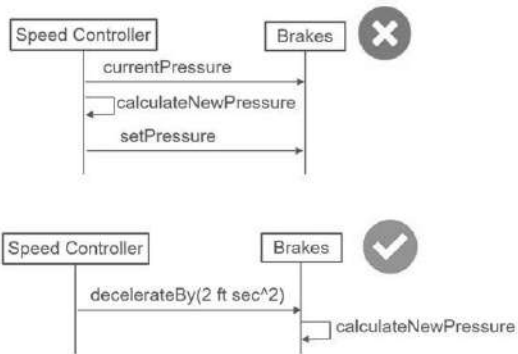
SCALED AGILE® © Scaled Agile, Inc.

299

Notes:

Favor assertive code over inquisitive code

- ▶ Inquisitive: Asks another object for information and does computation with it
 - Calculations embedded in client
- ▶ Assertive: Asks another object for result of computation
 - Object with the data performs the calculations



```
sequenceDiagram
    participant SC as Speed Controller
    participant B as Brakes
    SC->>B: currentPressure
    B-->>SC: 
    SC->>B: calculateNewPressure
    B-->>SC: 
    SC->>B: setPressure
    B-->>SC: 
    Note over SC,B: Inquisitive (X)

    SC->>B: decelerateBy(2 ft sec^2)
    B->>B: calculateNewPressure
    B-->>SC: 
    Note over SC,B: Assertive (✓)
```

SCALED AGILE® © Scaled Agile, Inc.

37

Notes:

Low-complexity code improves quality

```
if (a) {  
    if (b) {  
        if (c) {  
            if (d) {
```



```
method() {  
    // One hundred lines  
    ..  
    ..  
    ..  
}
```

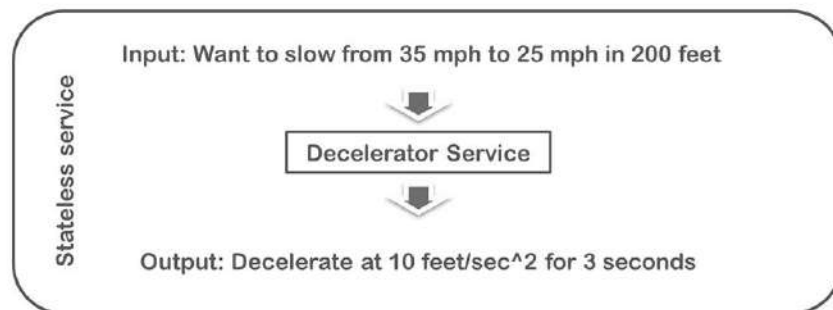


How long should a method be? How complex should a method be?

Notes:

Stateless services are easier to test

- ▶ Stateless services require no initialization
 - The output is dependent on the current input, not prior input
- ▶ Thus they are easier to test and re-use



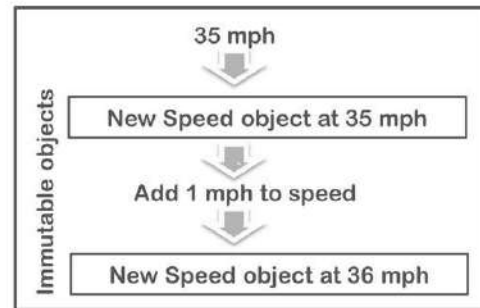
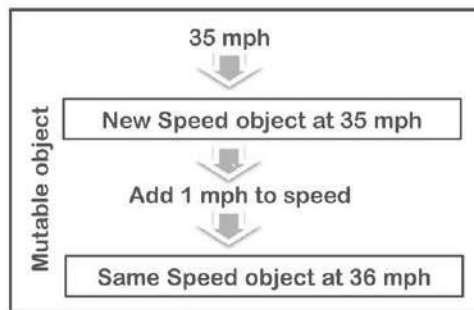
SCALED AGILE[®] © Scaled Agile, Inc.

39

Notes:

Immutable objects are easier to test

- ▶ Mutable entities have operations that can change their state
- ▶ Immutable entities have their state set once (at creation)
 - Immutable entities are usually easier to test



SCALED AGILE® © Scaled Agile, Inc.

40

Notes:

Handle errors consistently

- ▶ Consistency is simplicity
- ▶ Decide on a consistent strategy for dealing with errors
 - Consider failure to be an expectation, not an exception
 - Decide who should handle errors: client, implementation, or some higher entity
 - Log failures in a consistent manner
 - In general, do not use NULL/null as a return value indicating an error



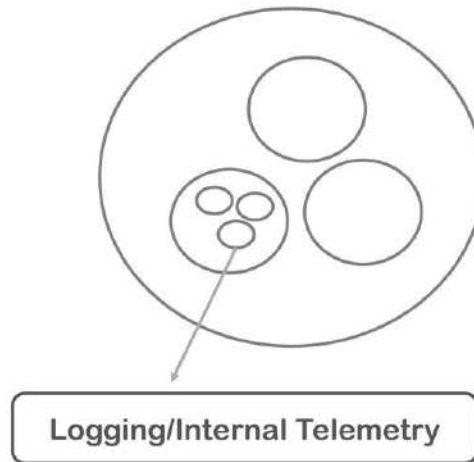
SCALED AGILE® © Scaled Agile, Inc.

41


Notes:

Log events in a consistent manner

- ▶ Define interesting test points to help determine system behavior during execution (internal telemetry)
- ▶ Decide on the important things to log
 - Logging everything can be overwhelming
 - Use log levels that can be controlled dynamically to vary granularity
- ▶ Plan your logging scheme so that it's testable



Notes:



Discussion: Code quality


Discuss

10 min

Share

5 min

- **Step 1:** Individually, consider examples of your own code or use the example provided in the workbook
- **Step 2:** At your table, discuss how you would improve the code quality
- **Step 3:** Share how you track code quality



SCALED AGILE® © Scaled Agile, Inc.

43

Notes:

9.4 Explain the benefits of collective ownership

SCALED AGILE® © Scaled Agile, Inc.

44

Notes:

Coding standards allow for collective ownership


- ▶ Code should be owned by the team, not the individual
- ▶ Coding standards allow for easier readability/maintainability
- ▶ Peer review helps maintain code to a standard
 - Pair programming/pair work fosters continuous review
 - Programming with non-programmers can help readability
- ▶ Pair/group design results in high-quality designs

SCALED AGILE® © Scaled Agile, Inc.


45


Notes:

9.4 Explain the benefits of collective ownership



Activity: What are the benefits of code standards?





Our stakeholders are...

We think our coding standards are good because...

Who are the stakeholders of your code?

How can you test that you have good coding standards?

SCALED AGILE® © Scaled Agile, Inc.

46

Notes:



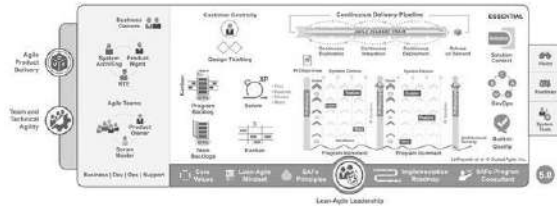
Discussion: Code quality in SAFe

Discuss



► Discuss:

- Which SAFe roles are responsible for ensuring good code quality?
- When in SAFe would we apply the coding practices discussed here?



SCALED AGILE® © Scaled Agile, Inc.

48

Notes:



Agile Software Engineering Action Plan



- **Step 1:** Locate the Agile Software Engineer Action Plan at the end of your workbook
- **Step 2:** Considering what we have covered, write down at least one improvement item in your Agile Software Engineer Action Plan
- **Step 3:** Share the item you wrote with the class.



Notes:



Notes:

Lesson Review

In this lesson, you:

- 9.1 Identify code qualities
- 9.2 Describe cohesion and coupling
- 9.3 Describe other code qualities
- 9.4 Explain the benefits of collective ownership

Notes:

Lesson 10

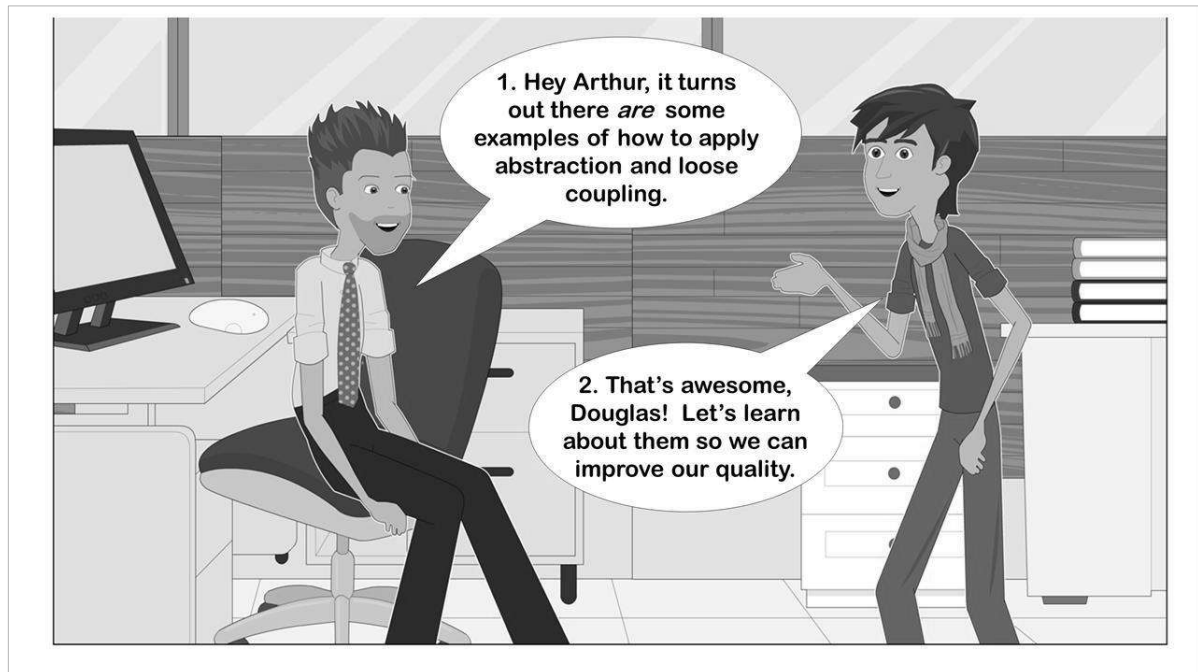
Building Systems with Design Quality

Learning Objectives:

- 10.1 Explore design tradeoffs
- 10.2 Explain Interface-Oriented Design
- 10.3 Apply quality decomposition practices
- 10.4 Apply differentiation and synthesis
- 10.5 Apply software design patterns



SAFe® Authorized Course Attending this course gives students access to the SAFe® Agile Software Engineer exam and related preparation materials.



Notes:

10.1 Explore design tradeoffs

SCALED AGILE © Scaled Agile, Inc.

4

Notes:

The rule of three

If you cannot come up with three solutions to a problem, you do not understand the problem.

—Gerald Weinberg



Example: Speed limit object, accelerator object

- Speed limit object tells accelerator whether to speed up
- Accelerator asks speed limit object what the current speed is
- Speed controller (third object) asks speed limit object for speed, tells accelerator whether to speed up

SCALED AGILE © Scaled Agile, Inc.

5

Notes:

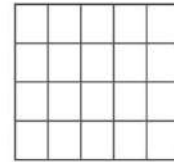


Discussion: The spreadsheet conundrum



Consider if you have to transfer a spreadsheet of cells onto a device (e.g. a file) and then transfer from the device back into a spreadsheet.

- ▶ Step 1: Using a flip chart sheet, illustrate three ways you could arrange the data on the device.
- ▶ Step 2: Explain the advantages and disadvantages of each.
 - What are the consequences of using each solution?
 - What in the problem space could possibly help you determine the best solution?



SCALED AGILE® © Scaled Agile, Inc.

6

Notes:

Determine criteria for choosing design alternatives

- ▶ Forces are criteria used to justify designs and implementations.
- ▶ Explore multiple approaches to determine the solution (set-based design).
- ▶ Then pick one approach, based on:
 - The forces in the problem (contextual forces)
 - The implementation forces (how it will be coded—will it be easier or harder)
 - The consequential forces (what changes will be easier and which will be harder)

SCALED AGILE® © Scaled Agile, Inc.

7


Notes:

10.2 Explain Interface-Oriented Design

SCALED AGILE © Scaled Agile, Inc.

8

Notes:



Discussion: Interfaces

Discuss
5 min

Share
5 min

- ▶ **Step 1:** Recall interfaces from your current system, or systems you have designed in the past.
- ▶ **Step 2:** Discuss the following:
 - Did those interfaces help in maintaining or deploying your system?
 - What are some practices or principles you employ in designing interfaces?

SCALED AGILE © Scaled Agile, Inc.

9

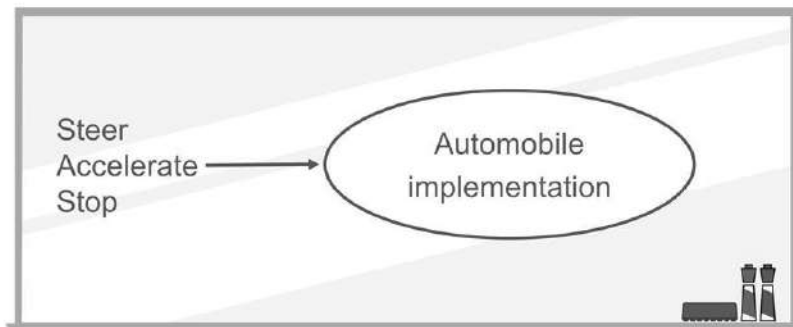
Notes:



Design to interfaces, not implementations

Notes:

Understand interfaces

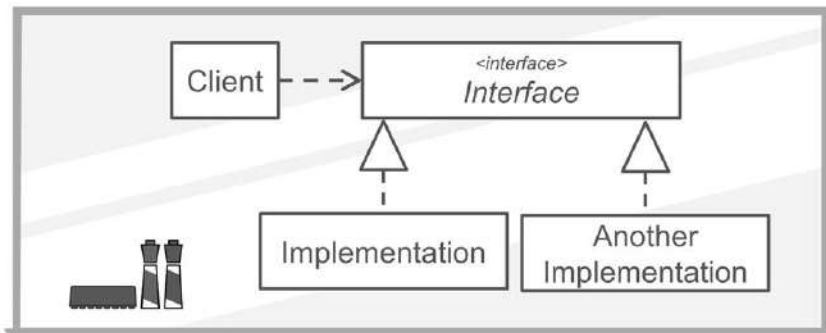


*How many different kinds of automobiles have you driven?
How many different interfaces did you need to learn?
Now, what about cruise control? Radio?*

Notes:

Design to interfaces, not implementations

- ▶ Interfaces act as contracts between implementations and clients
 - Interfaces define responsibilities of entity independent of its implementation
 - Interface remains consistent for both syntax and semantics



SCALED AGILE® © Scaled Agile, Inc.

12

Notes:

Interface-oriented design encompasses many principles

- ▶ Interface Separation Principle: (SOLID)
- ▶ Dependency inversion principle: (SOLID)
- ▶ Liskov substitution principle: (SOLID)



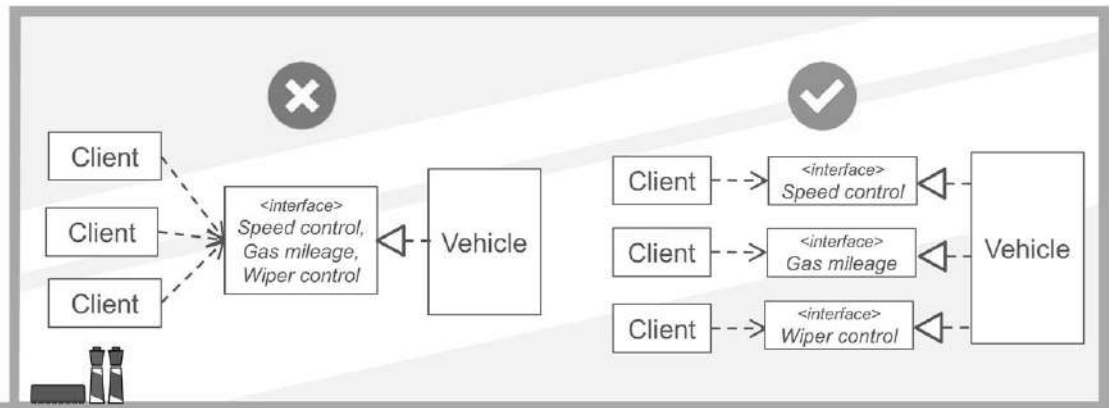
SCALED AGILE® © Scaled Agile, Inc.

13

Notes:

Interface separation principle (SOLID)

- Favor multiple, client-specific interfaces over one large, general-purpose interface. Build only what each client needs.



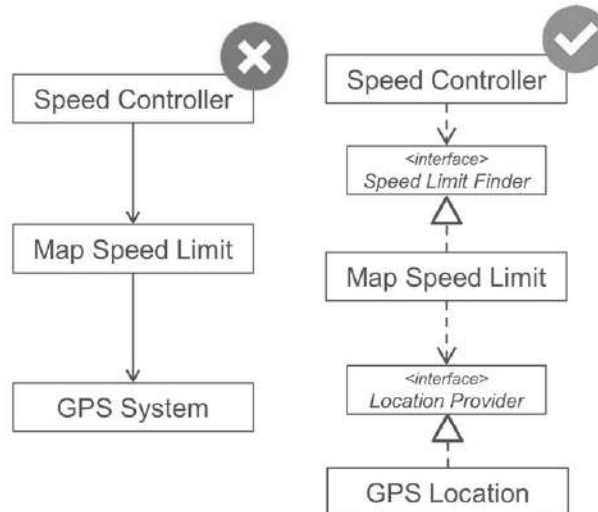
SCALED AGILE® © Scaled Agile, Inc.

14

Notes:

Use interfaces to eliminate Dependency Inversion (SOLID)

- High-level modules should not depend upon low-level modules.
 - Both should depend upon abstractions.



SCALED AGILE® © Scaled Agile, Inc.

15

Notes:

Liskov substitution principle (SOLID)

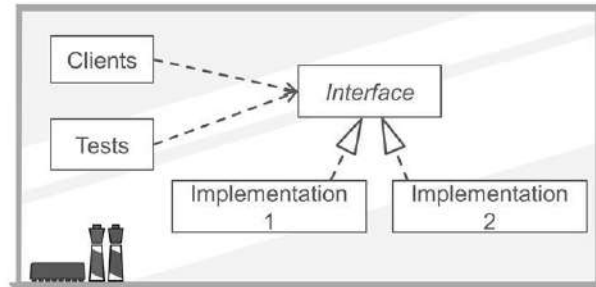
Let $p(x)$ be a property provable about objects x of type T . Then $p(y)$ should be true for objects y of type S where S is a subtype of T .

— Barbara Liskov

Clients of an interface have no dependency on the implementation and may use any implementation without knowing.

— Bob Martin

- ▶ Contract specifies the interface's behavior:
Given/When/Then
- ▶ Tests are also tied to interface, not implementation
- ▶ All implementations must pass same test



SCALED AGILE[®] © Scaled Agile, Inc.

16

Notes:

Three laws of interfaces

1. An implementation of an interface shall do what the interface defines:
 - Liskov substitution principle
 - *Design by Contract* by Bertrand Meyer
2. An implementation shall do no harm (use excessive memory, hold locks, etc.)
3. If an implementation is unable to perform its responsibilities, it shall notify someone.



SCALED AGILE[®] © Scaled Agile, Inc.

17

Notes:

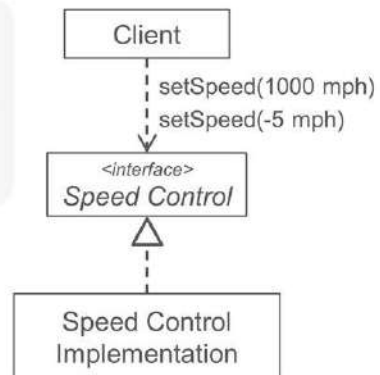
Using *Design by Contract*'s pre- and post-conditions



Example for `setSpeed()`

- Pre-condition: argument in the range 0..90 mph
- Post-condition: speed is set

- ▶ If pre-condition is not true, contract may not be satisfied and result is indeterminate
- ▶ Tests should exercise all combinations

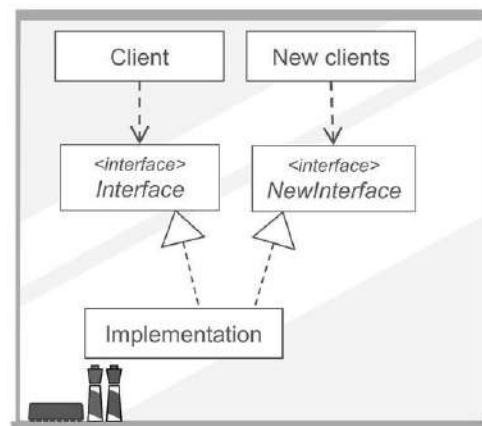


In each interface method, should you validate that the input matches the contract or should client validate the input? What are the tradeoffs?

Notes:

New interfaces for new semantics

- ▶ Adding new methods to an interface typically does not require existing clients to change
- ▶ Create a new interface if interface's semantics change
- ▶ Plan for moving clients from the existing interface to the new one



Notes:

Test for interface usability

- ▶ Interfaces should be tested for usability
- ▶ Tests are the first client of an interface
- ▶ Tests can also serve as documentation for the interface
- ▶ If a test is difficult to understand, perhaps the interface is hard to use




Notes:

10.3 Apply quality decomposition practices


SCALED AGILE © Scaled Agile, Inc.


21

Notes:




Activity: Decomposing entities





We decompose entities by...

What are ways you decompose entities today?





22

Notes:

Ways of expressing separation

- ▶ 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's easier to lump a splitter than to split a lump






23

Notes:


Programming by intention

- Supports understanding and reuse
- Reflects the developer's thinking at the appropriate level of abstraction
- Creates more modular and reusable code by separating the concerns for intention

 `controlSpeed() {
 // code to get speed limit

 // code to change speed

 // code to maintain speed
}`

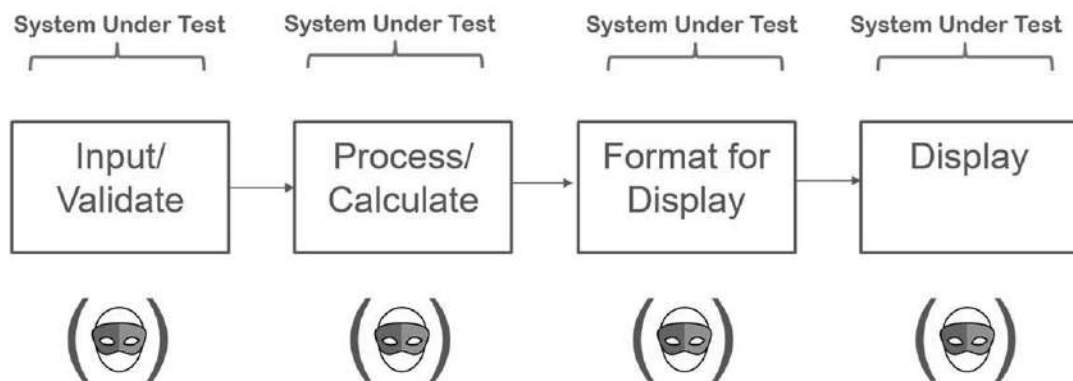
 `controlSpeed() {
 speed = getSpeedLimit()
 changeSpeed(speed)
 maintainSpeed()
}
getSpeedLimit() {
}
changeSpeed(speed) {
}
maintainSpeed() {
}`

SCALED AGILE[®] © Scaled Agile, Inc.

24

Notes:

Programming by intention supports internal testing



Note: Use test doubles, if necessary, to independently tests internal parts

SCALED AGILE[®] © Scaled Agile, Inc.

25

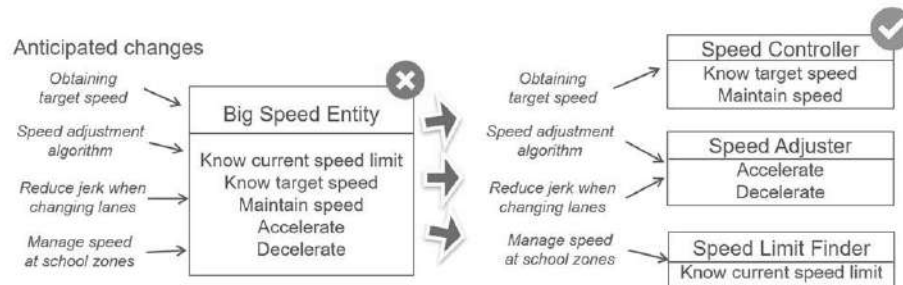
Notes:

Separate concerns with Single Responsibility principle (SOLID)

Each software module should have one and only one reason to change.

Gather together the things that change for the same reasons. Separate those things that change for different reasons.

—Bob Martin

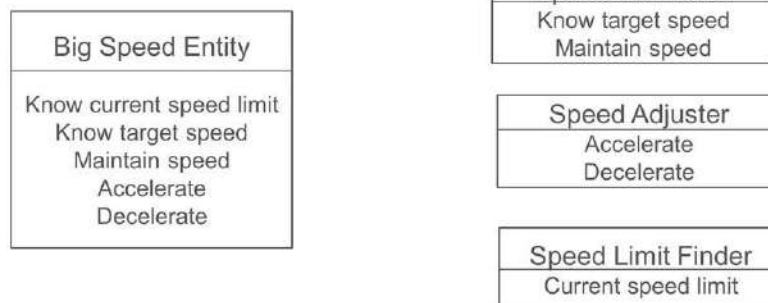


SCALED AGILE[®] © Scaled Agile, Inc.

26

Notes:

Good separation of concerns simplifies reuse



Another component needs to smooth speed for braking at stop signs and traffic lights. Which design provides simpler reuse? Why?

SCALED AGILE[®] © Scaled Agile, Inc.

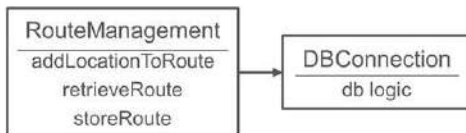
27

Notes:

Separation example: Database access

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

Separation of concerns delegates retrieval, storage, and other DB activities to another object, DBConnection



```

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
    }
}
  
```

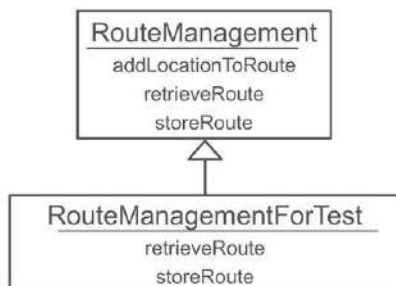
SCALED AGILE[®] © Scaled Agile, Inc.

28

Notes:

Separation creates **seams** for test doubles

Overridden methods are test doubles that provide alternative storage and retrieval logic for testing



```

class RouteManagementForTest
    extends RouteManagement {
    Route retrieveRoute(RouteID id) {
        // return data for test
        return new Route(id, ....)
    }

    storeRoute(Route route) {
        // check for correct data in route
        assert(...)
    }
}
  
```

SCALED AGILE[®] © Scaled Agile, Inc.

29

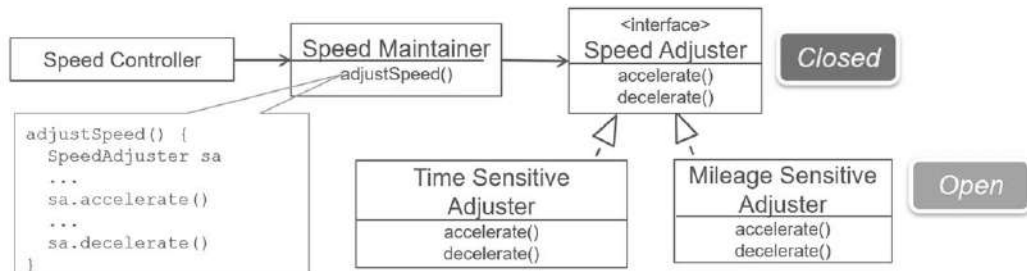
Notes:

Create good structures with Open/Closed principle (SOLID)

Software entities should be open for extension, but closed for modification.
— Bertrand Meyer

- Keep current entities intact while extending them to support new behavior

! Interface Open/Closed

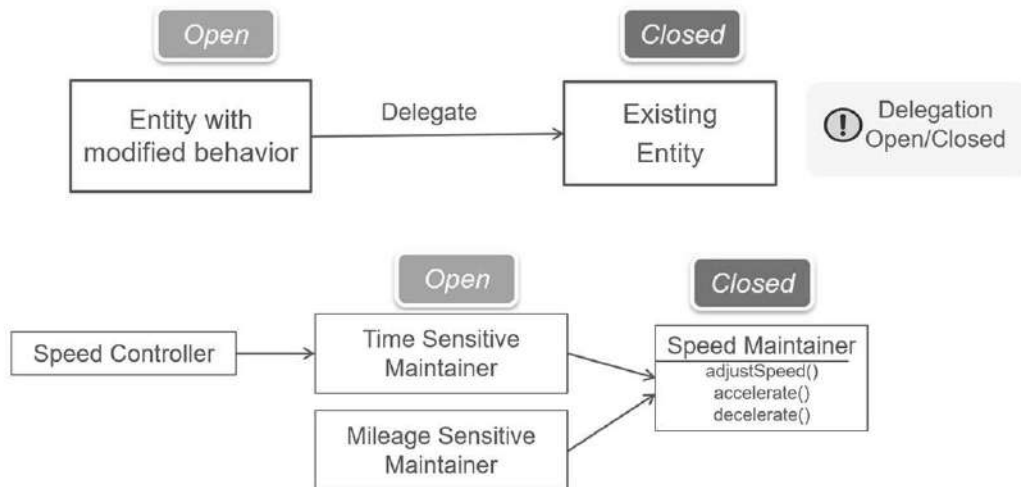


SCALED AGILE® © Scaled Agile, Inc.

30

Notes:

Delegation is another application of Open/Closed principle



SCALED AGILE® © Scaled Agile, Inc.

31

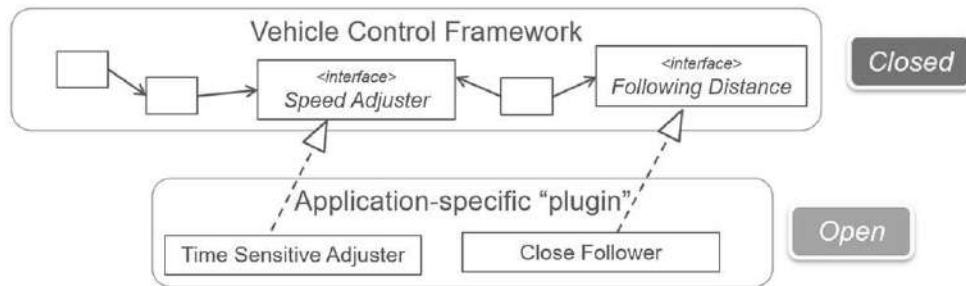
Notes:

Scale Open/Closed to large systems and frameworks

Plugin systems are the ultimate consummation, the apotheosis, of the Open-Closed Principle
— Bob Martin

- Scales extension to large systems (e.g., Eclipse, ASP.NET, Android OS)

! 'Plugin'
Open/Closed



SCALED AGILE® © Scaled Agile, Inc.

32

Notes:



Discussion: Design principles and practices



- ▶ **Step 1:** Identify entities from your domain that could meet the principles and practices discussed:
 - Open to extension, closed to modification
 - Ability to easily substitute new behavior
 - Have a single responsibility
- ▶ **Step 2:** Share with the class.


Notes:

10.4 Apply differentiation and synthesis

SCALED AGILE © Scaled Agile, Inc.

34

Notes:



Video discussion: How do we design?

Watch
1 min

Share
3 min

HOW DO WE DESIGN?

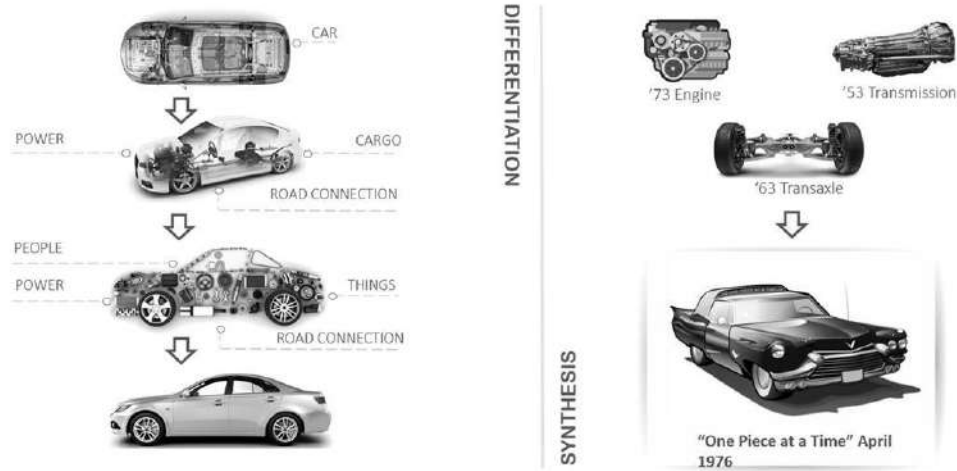
Differentiation or Synthesis?

SCALED AGILE © Scaled Agile, Inc.

35

Notes:

Contextual design: Differentiation versus synthesis



SCALED AGILE® © Scaled Agile, Inc.

36

Notes:

Differentiation and synthesis: A combination

- At some point, you reach pre-defined components
- Knowledge of components helps in the differentiation




SCALED AGILE® © Scaled Agile, Inc.

37

Notes:

A software example of differentiation versus synthesis

Differentiation



Screen

Main topics

Menu


First

Second

Standard Item

Special notices

Synthesis



Form Entry

Data entry, e.g. text field

Button

NOTE: Buttons and text entry fields are existing components


SCALED AGILE © Scaled Agile, Inc.

38

Notes:

Activity: How to find components/classes in libraries

Shout 3 min



We find components and classes by ...

Do you have existing components/classes?

How do you (as a developer) find components/classes in those libraries?

AGILE © Scaled Agile, Inc.

39

Notes:

10.5 Apply software design patterns

SCALED AGILE © Scaled Agile, Inc.

40

Notes:



Discussion: Design patterns





SCALED AGILE® © Scaled Agile, Inc.

41

Notes:

Apply design patterns

- ▶ *Design Patterns* by Gang of Four (GoF) popularized the patterns movement
 - Provided 24 reusable elements
 - Served as examples of good design principles and practices
- ▶ Patterns are conceptual, descriptions of something that has worked well in the past
 - Patterns are not a specific implementation



Notes:

Why do we need design patterns?

- ▶ Improve quality by adding flexibility to a system
 - Make extensions easier to add to new implementations
 - Simplify testing to easily inject test doubles
- ▶ The names of patterns can become design domain terms
 - Improve readability and maintenance



Notes:

Common design patterns

- ▶ Strategy pattern
- ▶ Factory pattern
- ▶ Dependency injection
- ▶ Adapter pattern
- ▶ Mediator pattern
- ▶ Proxy/decorator pattern

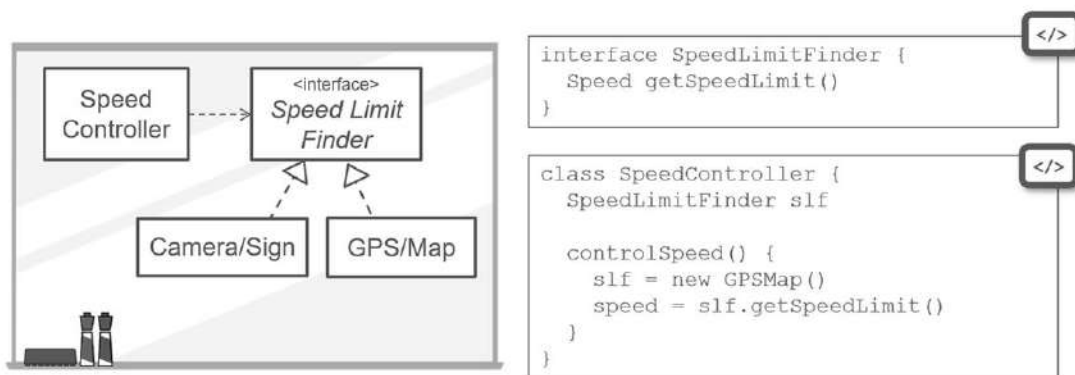
SCALED AGILE® © Scaled Agile, Inc.

44

Notes:

Implementing strategy patterns

Class delegates to interface rather than implementation.



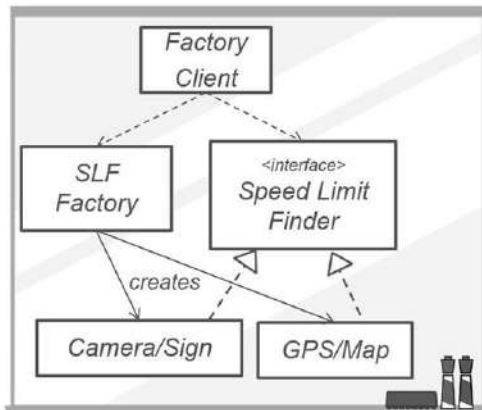
SCALED AGILE® © Scaled Agile, Inc.

45

Notes:

Use factory patterns to separate creation from usage

Also easy to inject test doubles



✗

```
class OldClient {
    SpeedLimitFinder slf

    // Tight coupling
    slf = new GPSMap()
    slf.getSpeedLimit()
}
```

✓

```
class FactoryClient {
    SpeedLimitFinder slf

    slf = SpeedLimitFinderFactory.getSLF()
    slf.getSpeedLimit()
}
```

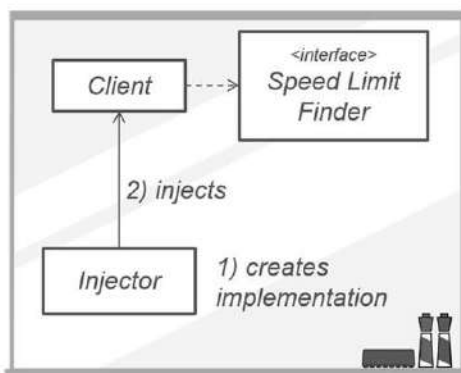
SCALED AGILE[®] © Scaled Agile, Inc.

46

Notes:

Dependency injection provides inversion of control

Class dependencies configured at runtime (configuration replaces the factory)



Explicit injection point

</>

```
class Client {
    @inject
    SpeedLimitFinder slf
    // created by injection

    slf.getSpeedLimit()
}
```

Injector configuration

</>

```
<interface id = "SpeedLimitFinder"
  class = "com.scaledagile.GPSMapper">
</interface>
```

SCALED AGILE[®] © Scaled Agile, Inc.

47

Notes:

Use factories to avoid redundancy (1)

```

accelerate() {
  switch(value) {
    case MILEAGE_SENSITIVE:
      ms_accelerate()
    case TIME_SENSITIVE:
      ts_accelerate()

    // and more
  }
}

```



```

decelerate() {
  switch(value) {
    case MILEAGE_SENSITIVE:
      ms_decelerate()
    case TIME_SENSITIVE:
      ts_decelerate()

    // and more
  }
}

```



Guideline: If thinking about copying and pasting a switch statement, think harder about creating a strategy with a factory that contains that switch statement.

Notes:

Use factories to avoid redundancy (2)

```

interface JerkControl {
  accelerate()
  decelerate()
}

class JerkControlFactory {
  JerkControl getJC(){
    switch(value) {
      case MILEAGE_SENSITIVE:
        return new MileageSensitive
      case TIME_SENSITIVE:
        return new TimeSensitive

      // and more
    }
  }
}

```



```

class MileageSensitive
  implements JerkControl {
  accelerate() {
    ms_accelerate()
  }
  decelerate() {
    ms_decelerate()
  }
}

```

```

class TimeSensitive
  implements JerkControl {
  accelerate() {
    ts_accelerate()
  }
  decelerate() {
    ts_decelerate()
  }
}

```

Notes:

Alternative designs make different changes easier

Original version

	TIME_SENSITIVE	MILEAGE_SENSITIVE
accelerate		
decelerate		

Another
switch
statement

Factory version

	TIME_SENSITIVE	MILEAGE_SENSITIVE
accelerate		
decelerate		

Another
object

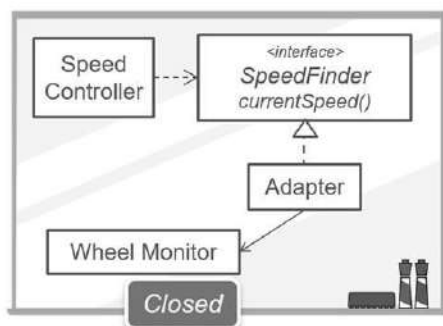


What changes are easier in each version?

Notes:

Adapter pattern adapts one interface to another

- ▶ Can transform mismatching methods, arguments, paradigms
- ▶ 'Opens' an otherwise closed entity for another use



```

class WheelMonitor {
    int speed() {}
}
    
```

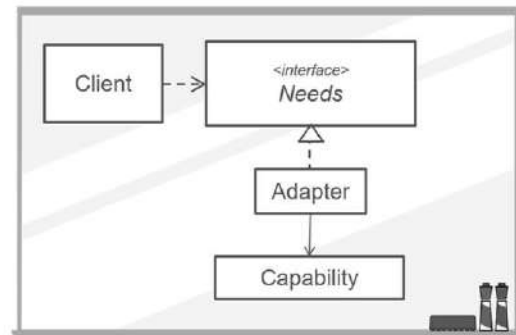
```

// Adapter provides speed finder
interface
class Adapter implements SpeedFinder {
    Speed currentSpeed() {
        int speedInt = WheelMonitor.speed()
        return new Speed(speedInt)
    }
}
    
```

Notes:

Adapters support interface segregation

- ▶ Adapter pattern supports the interface segregation principle (client-specific interfaces)
- ▶ Separates needs from capabilities



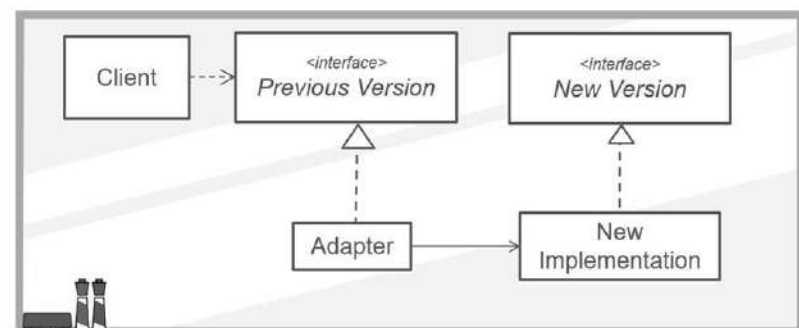
SCALED AGILE[®] © Scaled Agile, Inc.

52

Notes:

Adapter support versioning

- ▶ Adapter patterns allow for versioning of application program interfaces (APIs)
 - Eventually deprecate the old version



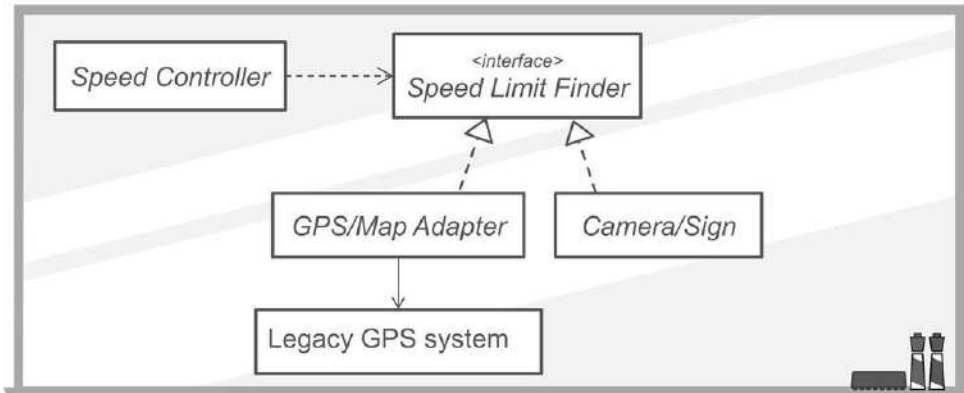
SCALED AGILE[®] © Scaled Agile, Inc.

53

Notes:

Systems will have many combinations of patterns

Strategy, adapter are shown



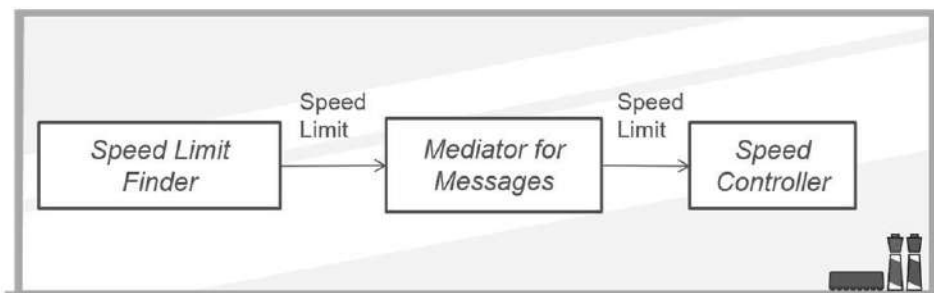
SCALED AGILE[®] © Scaled Agile, Inc.

54

Notes:

Mediator pattern

- ▶ Intermediate entity handles delivery of message
- ▶ Decouples sender and receiver and removes dependency on one another
- ▶ Simplifies testing by applying test double to sender or receiver



SCALED AGILE[®] © Scaled Agile, Inc.

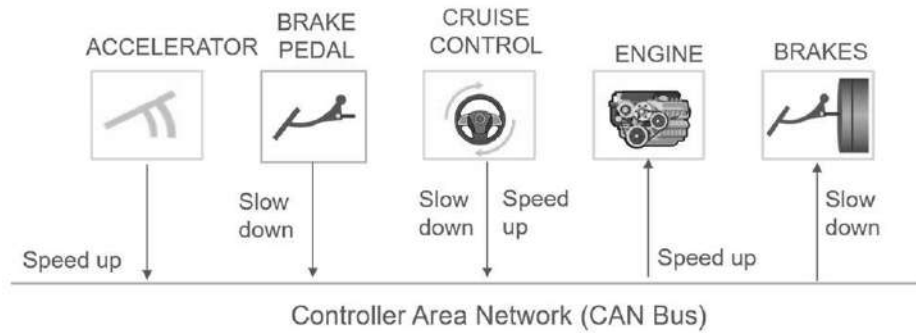
55

Notes:

Mediator pattern: Example

- Patterns such as the mediator can appear in both:

- Small-scale design
- Large-scale architecture



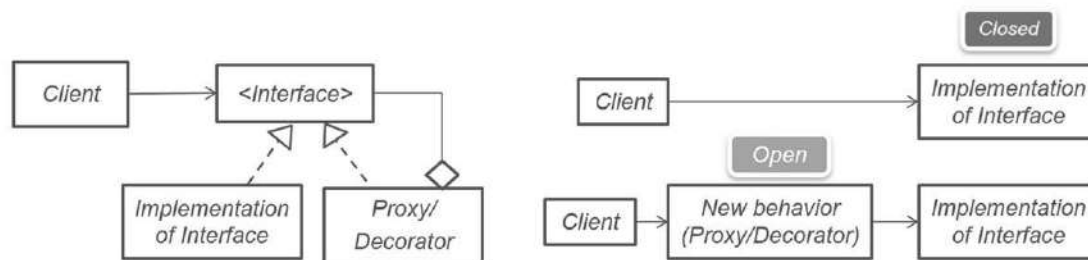
SCALED AGILE[®] © Scaled Agile, Inc.

56

Notes:

Proxy/decorator pattern

- Adds behavior to an entity without affecting clients or implementations
- Determines additional responsibilities dynamically
- Supports the single responsibility principle



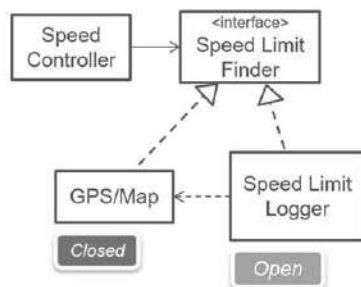
SCALED AGILE[®] © Scaled Agile, Inc.

57

Notes:

Proxy/decorator for speed limit finder: Example

Example: Add a logger between speed controller and GPS/Map speed limit finder



```

interface SpeedLimitFinder {
    Speed getSpeedLimit()
}

class SpeedLimitLogger
    implements SpeedLimitFinder {
    Speed getSpeedLimit() {
        speed = GPSMap.getSpeedLimit()
        aLogger.log("Speed limit is", speed)
        return speed
    }
}

class SpeedController {
    controlSpeed() {
        SpeedLimitFinder slf = SLFFactory.getSLF()
        // gets the SpeedLimitLogger
        speed = slf.getSpeedLimit()
    }
}
  
```

Factory hides the proxy

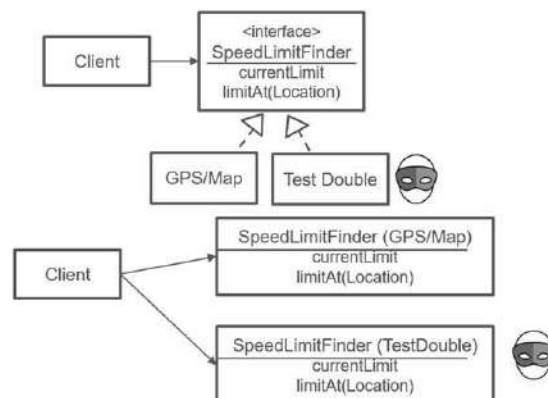
SCALED AGILE® © Scaled Agile, Inc.

58

Notes:

Design patterns help create seams

- ▶ Seams are locations in code where test doubles can be injected
- ▶ Seams can also come from compile-time (exe) or run-time (jar, dll) bindings
 - Two implementations of SpeedLimitFinder are in different directories




What are others examples in your code of 'seams' to inject testing?

SCALED AGILE® © Scaled Agile, Inc.

59

Notes:



Activity: Which pattern to choose


Prepare
10 min

Gallery Walk
8 min

- ▶ **Step 1:** With your team, determine if any of the patterns apply to your story.
- ▶ **Step 2:** Create a model of the pattern on a flip chart sheet or a whiteboard.
- ▶ **Step 3:** If your previous design exercise outputs are less readable, make them readable.
- ▶ **Step 4:** Do a gallery walk:
 - One person stays with each team's design that has been developed over the past several exercises.
 - Other team members walk around, look at other designs, and ask questions of the team member.

SCALED AGILE® © Scaled Agile, Inc. 60

Notes:




Activity: Build your Agile Software Engineer Plan

Share
2 min

- ▶ **Step 1:** Considering what you have learned, write down at least one improvement item in your *Agile Software Engineer Plan*
- ▶ **Step 2:** Share the item you wrote with the class

In this lesson, we:

- ▶ Explored design tradeoffs
- ▶ Explained interface-oriented design
- ▶ Applied quality decomposition practices
- ▶ Applied differentiation and synthesis
- ▶ Applied software design patterns



SCALED AGILE® © Scaled Agile, Inc.

61

Notes:



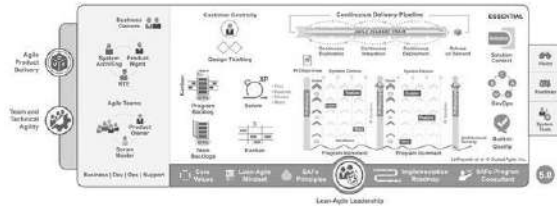
Discussion: Building systems with design quality in SAFe

Discuss



► Discuss:

- What roles in SAFe are responsible for applying design quality?
- When would you perform the practices discussed in this section?



DISCUSS



SCALED AGILE® © Scaled Agile, Inc.

62

Notes:



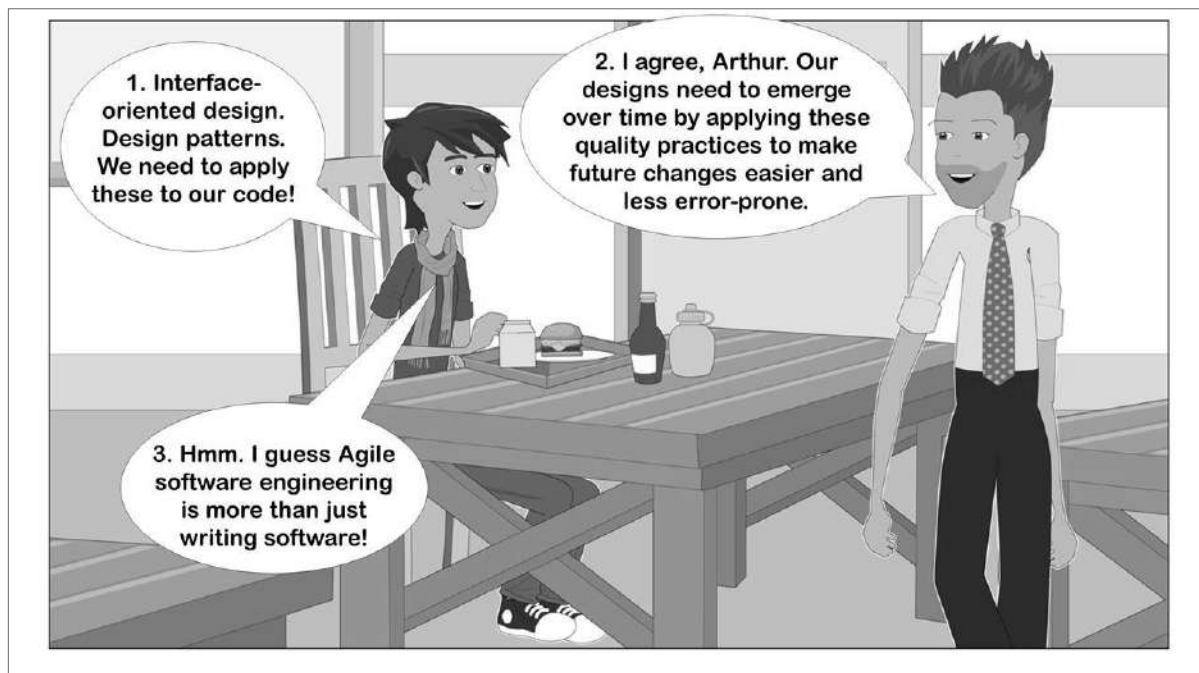
Agile Software Engineering Action Plan



- ▶ **Step 1:** Locate the Agile Software Engineer Action Plan at the end of your workbook
- ▶ **Step 2:** Considering what we have covered, write down at least one improvement item in your Agile Software Engineer Action Plan
- ▶ **Step 3:** Share the item you wrote with the class.



Notes:



Notes:

Lesson Review

In this lesson, you:

- 10.1 Explore design tradeoffs
- 10.2 Explain interface-oriented design
- 10.3 Apply quality decomposition practices
- 10.4 Apply differentiation and synthesis
- 10.5 Apply software design patterns

Notes:

Lesson 11

Implementing with Quality

Learning Objectives:

- 11.1 Design with tests
- 11.2 Apply test-driven development (TDD) practices
- 11.3 Implement test doubles and test data
- 11.4 Refactor to support new behavior of the code
- 11.5 Practice emergent design



SAFe® Authorized Course Attending this course gives students access to the SAFe® Agile Software Engineer exam and related preparation materials.



Notes:

11.1 Design with tests

SCALED AGILE © Scaled Agile, Inc.

4

Notes:



Discussion: Applying test-driven design

Discuss



Share



- ▶ **Step 1:** At your table, discuss the following about TDD:
 - Are you applying TDD? How is it working for you?
 - What are the units you are testing (module, class, method)?
 - Some call TDD test-driven development. Others call it test-driven design. What might be the difference?
- ▶ **Step 2:** Share with the class.

SCALED AGILE® © Scaled Agile, Inc.

5

Notes:

Design allocates responsibilities to entities

- ▶ Design allocates responsibilities for passing external tests to internal entities (components, classes, etc.)
- ▶ Design thinking uses the principles, practices, and qualities discussed in earlier lessons

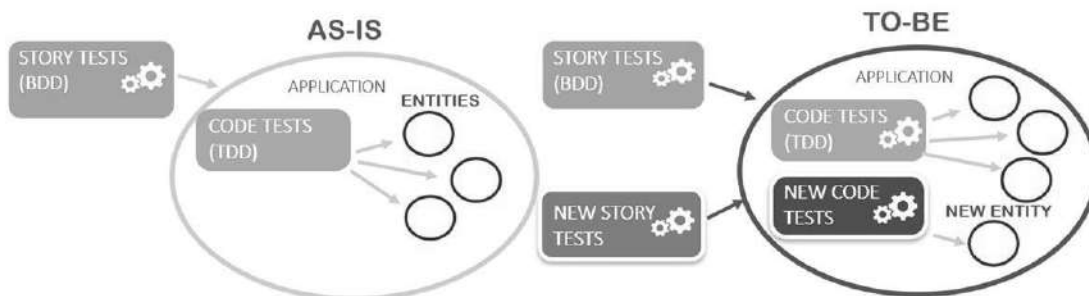


This is one approach to thinking about design. What are yours?

Notes:

Looking at tests from the context

New requirements mean new (or modified) entities with new tests.



Notes:

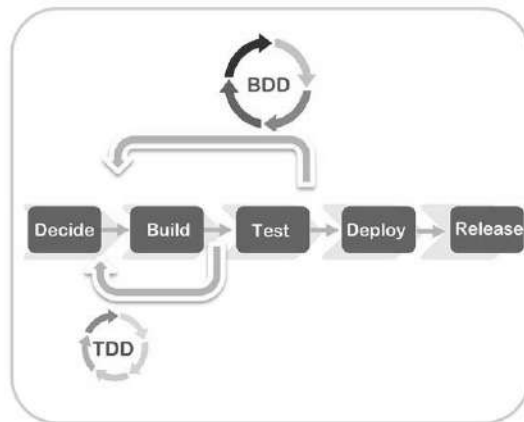
11.2 Apply test-driven development (TDD) practices

SCALED AGILE © Scaled Agile, Inc.

8

Notes:

Build quality into the Value Stream with BDD and TDD



SCALED AGILE © Scaled Agile, Inc.

9

Notes:

A BDD external test

User Story

As a driver, I want to enter a destination and get the set of route instructions to that destination



External BDD Test

```

Given automobile's location is 732 9th St
And automobile's heading is south
When the destination entered is 1802 W Main St
Then route instructions are
  Turn left in .15 miles onto Main St
  Stop in .2 miles
  
```

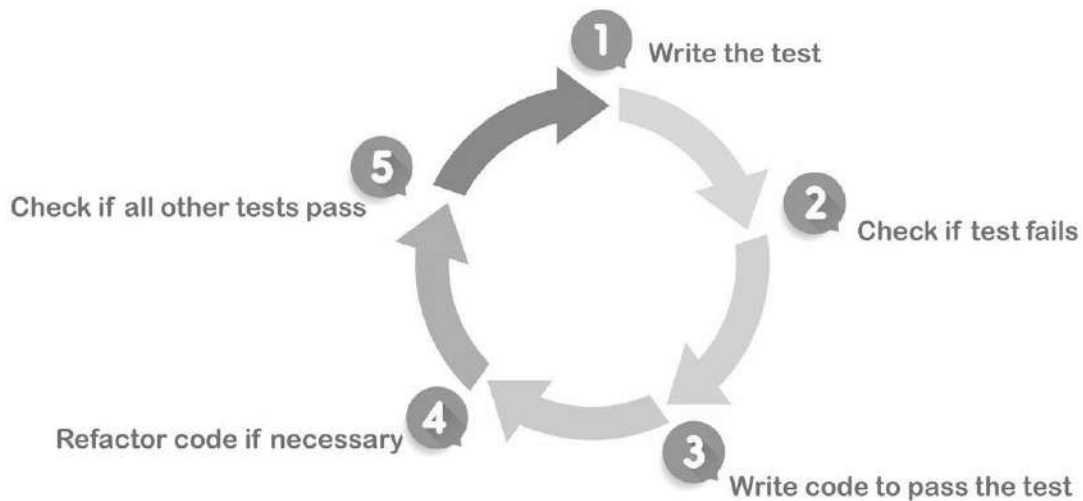


SCALED AGILE® © Scaled Agile, Inc.

10

Notes:

The test-driven development (TDD) cycle



SCALED AGILE® © Scaled Agile, Inc.

11

Notes:

Determine small internal test for route

- Unit testing should test the smallest possible behavior
- For example, could develop a routing algorithm one small step/test at a time

Start	End	Street	Distance
A	B	9 th Street	.05
B	C	9 th Street	.1
C	D	Main Street	.1
D	E	Main Street	.1
B	F	Perry	.11
D	F	Iredell St	.1

? What are some simple routes to test?



SCALED AGILE © Scaled Agile, Inc.

12

Notes:

Start with the smallest possible test, then work from there

Origin	Destination	Directions	Notes on behavior
A	B	Stop .05 mile	Single segment
A	C	Stop .15 miles	Two segments
A	F	Straight .05 miles Turn Left Perry Street Stop .11 mile	One turn after one segment
A	D	Straight .15 miles Turn Left Main Street Stop .1 mile	One turn after two segments
A	A	Stop	Already at destination



SCALED AGILE © Scaled Agile, Inc.

13

Notes:

TDD unit test #1

- Get route instructions for being at destination

```
@test
OriginAndDestinationTheSameShouldReturnStop() {
    origin = LOCATION_A
    destination = LOCATION_A
    RouteInstructions ri = Router.getRouteInstructions(origin, destination)
    assertEquals(STOP, ri.get(0))
}
```

Notes:

TDD unit test #2

- Get route instruction for one segment and stop

```
@test
OriginAtDestinationWithOneSegmentShouldReturnSingleRI () {
    origin = LOCATION_A
    destination = LOCATION_B
    RouteInstructions ri = Router.getRouteInstructions(origin, destination)
    RouteInstruction expected = new RouteInstruction(STRAIGHT, new Distance(.05))
    assertEquals(expected, ri.get(0))
    assertEquals(STOP, ri.get(1))
}
```

Notes:



Video: TDD in action - Prime Factors (Java)

Duration
1 min



<https://vimeo.com/374466447/20364439c1>

SCALED AGILE® © Scaled Agile, Inc.

390

Notes:


Design good quality TDD tests

- ▶ BDD tests may fail for multiple reasons since they are typically large scope
- ▶ TDD Tests
 - A test should fail for one reason
 - A test should fail for no other reason
 - No other test should fail for the same reason (non-redundant tests)
- ▶ Unit tests that pass specify the behavior of an entity
 - TDD is a great way to create those unit tests

SCALED AGILE® © Scaled Agile, Inc.

17

Notes:



Activity: Small, incremental, lower-level tests

Prepare

5 min

Share

5 min

- **Step 1:** With your team, choose one of the Stories from your own context (or the *Autonomous Vehicle Parking* domain).
- **Step 2:** What incremental tests could you write for that Story? Construct some lower-level tests for one of the scenarios you wrote, in the format of `Given/When/Then`.

SCALED AGILE © Scaled Agile, Inc.

18

Notes:

11.3 Implement test doubles and test data

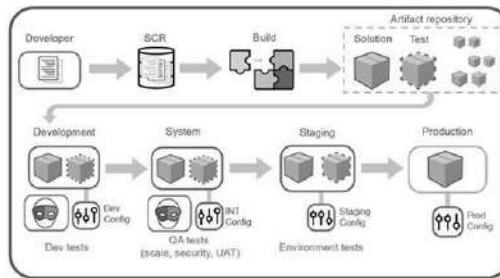
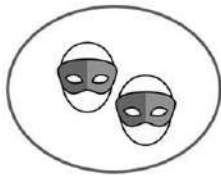
SCALED AGILE © Scaled Agile, Inc.

19

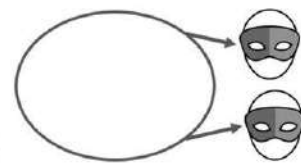
Notes:

From developer to production environments: Test doubles

Test doubles built into component (binary)



Test doubles as separately deployable services



*What are the benefits of each approach?
Where in our flow would we use one vs. the other? Why?*

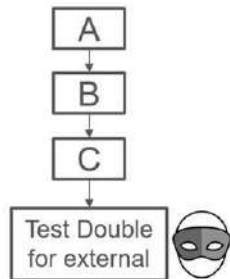
SCALED AGILE © Scaled Agile, Inc.

20

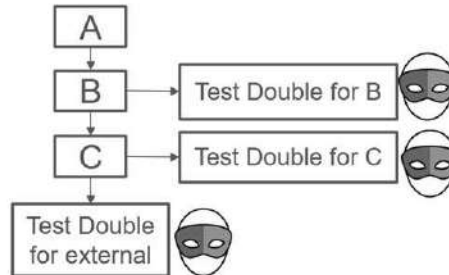
Notes:

Develop inside-out to create fewer test doubles

Inside out development



Outside in development



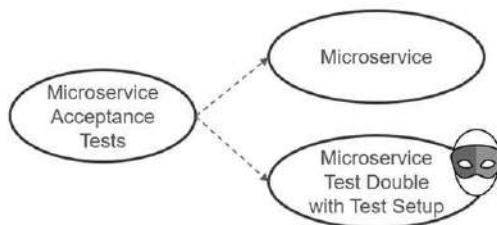
What are the implications of implementing A-B-C vs. C-B-A?

Notes:

Reusing services

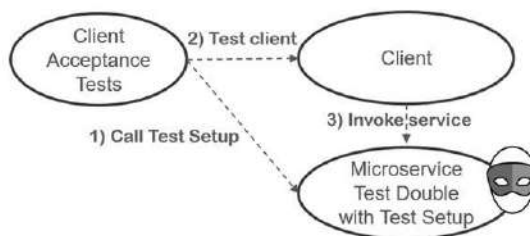
Service Creator should provide:

- Test double if service is expensive, slow, random, or non-idempotent
- Binary version of test double for fast testing



Service Clients should provide:

- Client acceptance test
- Input into acceptance tests for the service



Notes:

Reset state for flow test

```
Given speed is 0 mph  
When accelerating to 30 mph  
Then speed is achieved within 5 seconds
```

➔ To run test again, speed must be reset to 0



How can we do this?

Notes:

What to do about the data?

GPS/Mapper requires route data:


- For test repeatability, it should use data that does not change

Create separate tests for:

- Making sure new data is used
- Updating the data



Notes:



Discussion: Test data

Prepare
3 min

Share
2 min

- **Step 1:** Considering that databases are stateful (changes typically persist) and that repeatable tests typically need repeatable data, discuss the following:
 - How do you get repeatable data in your context?
 - How do you reset data in your context so that tests can be run again?
 - Should tests create their own data or should they use production data?
- **Step 2:** Share with the class

SCALED AGILE® © Scaled Agile, Inc.

25

Notes:

Strategies for creating repeatable test data

- ▶ Use a small test database that is restored after testing
- ▶ Store test databases in a cloud instance that is created when testing
- ▶ Use client's data and restore it after testing
- ▶ Use transactions on all database accesses, but do not commit in testing
- ▶ Use a proxy that caches database transactions

Notes:

11.4 Refactor to support new behavior of the code

SCALED AGILE® © Scaled Agile, Inc.

27

Notes:

Why refactor?


- ▶ Improve code quality to make code more maintainable
- ▶ Change code structure or design to support new requirements
- ▶ Enhance testability (create seams)



SCALED AGILE® © Scaled Agile, Inc.


28

Notes:



Activity: Refactorings?

Shout
3 min



I've refactored because....

What refactoring have you done?

AGILE © Scaled Agile, Inc.

29

Notes:

Refactoring does not change behavior

Legacy code is any code without tests (automated tests)

—Michael Feathers

- ▶ How to define behavior: the test
- ▶ How to ensure refactoring doesn't change behavior: the test
 - If no tests, write them first
- ▶ Refactoring should (usually) not require changes to test doubles
 - Test doubles that get called by the implementation reflect a constraint on the design

SCALED AGILE® © Scaled Agile, Inc.

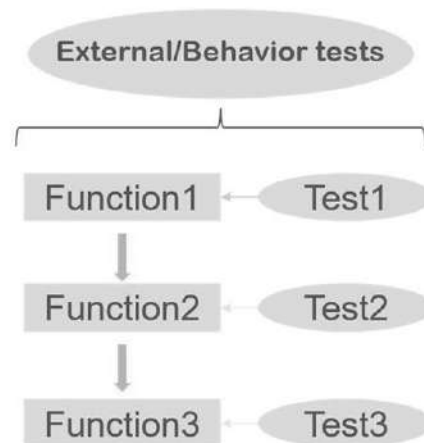
30

Notes:

Contrast refactoring vs restructuring

If we say we are refactoring, the system should always run.

- ▶ Refactoring:
 - Change in small steps
 - Never far from passing tests (BDD and TDD)
- ▶ Restructuring/redesign:
 - External acceptance tests ensure restructuring has not changed behavior
 - May not pass tests for a while
 - May need to also restructure tests



SCALED AGILE® © Scaled Agile, Inc.

31

Notes:

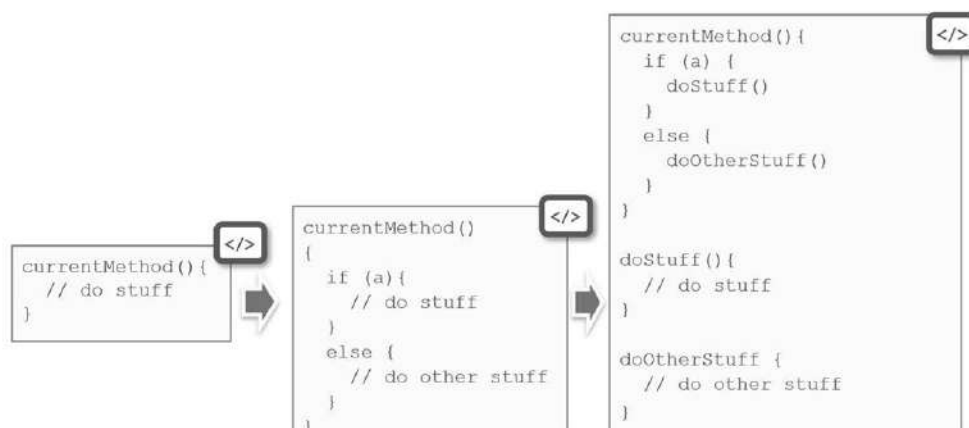
Common code refactoring needs

- ▶ Extract method to make them smaller, more cohesive
- ▶ Rename class, attributes, method to a more suitable name
- ▶ Move a method to a class that has the data (better cohesion)
- ▶ Extract an interface

Notes:

Refactor: Extract method example

As code evolves, the original structure becomes suboptimal.



Notes:

11.4 Refactor to support new behavior of the code

Example: Consider multiple alternatives before refactoring

```
someMethod(a,b,c) {  
  if (a) {  
    // Do stuff (lots of lines)  
    if (b) {  
      // Do more stuff (lots of lines)  
      if (c){  
        // Do still more stuff (lots of lines)  
      }  
    }  
  }  
}
```

</>



What is wrong with this code? How might you refactor it?

SCALED AGILE® © Scaled Agile, Inc.

34

Notes:

Refactoring: Extract method solution one

```
someMethod(a,b,c) {  
  if (a) {  
    // Do stuff (lots of lines)  
    checkB(b,c)  
  }  
}
```

</>

```
checkB(b,c) {  
  if (b) {  
    // Do more stuff (lots of lines)  
    checkC(c)  
  }  
}
```

</>

```
checkC(c) {  
  if (c){  
    //Do still more stuff (lots of lines)  
  }  
}
```

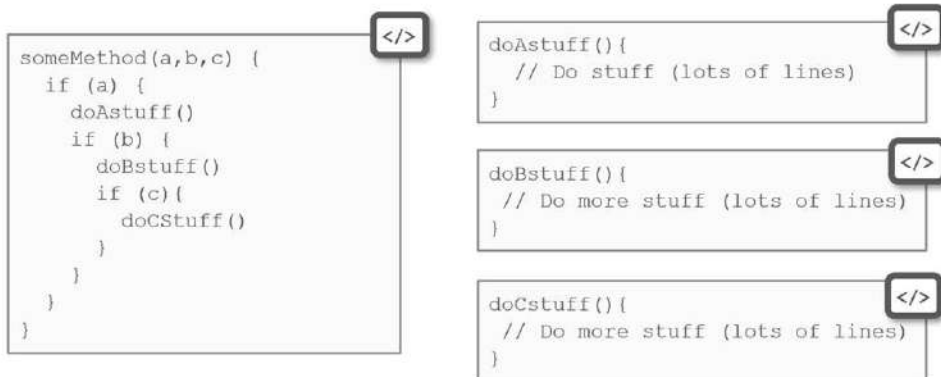
</>

SCALED AGILE® © Scaled Agile, Inc.

35

Notes:

Refactoring: Extract method solution two



Contrast the two refactoring solutions. What are the advantages and disadvantages of each?

Notes:

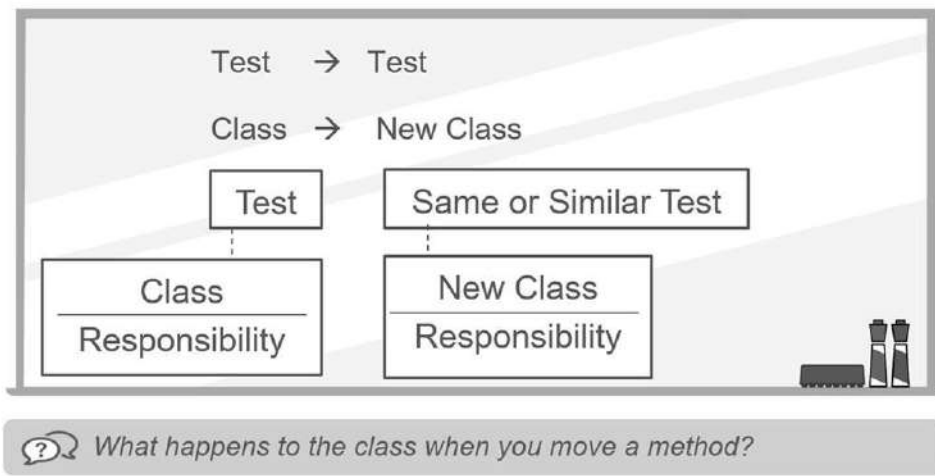
Renaming

- ▶ Makes code more readable
- ▶ Matches internal names to domain names



Notes:

Moving a method

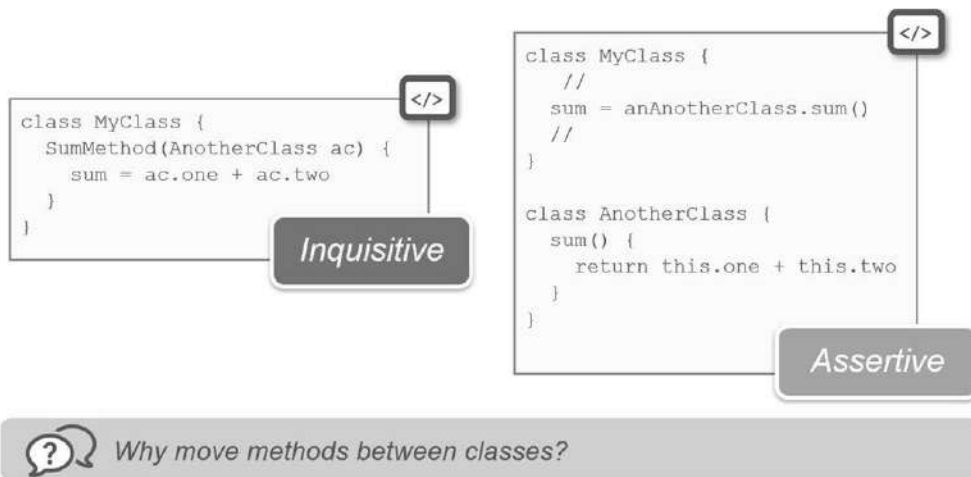


SCALED AGILE® © Scaled Agile, Inc.

38

Notes:

Moving a method: Example



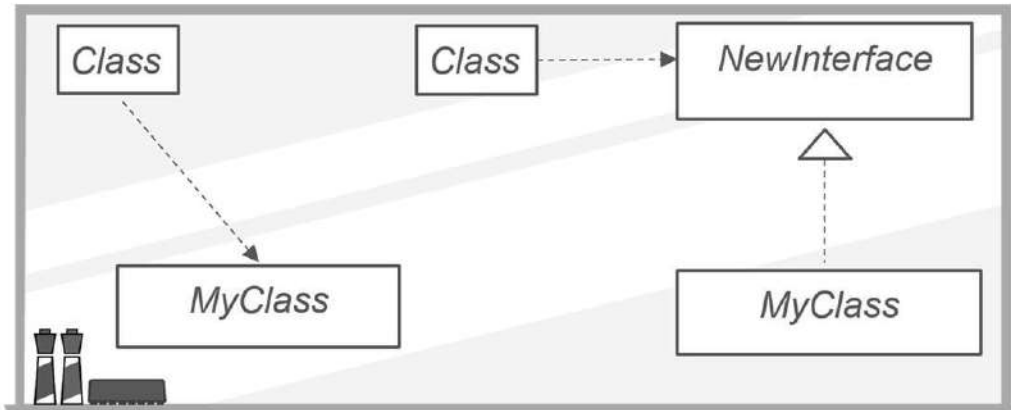
SCALED AGILE® © Scaled Agile, Inc.

39

Notes:

Extracting interface

Allows for patterns (e.g. strategy)



SCALED AGILE® © Scaled Agile, Inc.

40

Notes:

Extracting Interface: Example

```
class MyClass {  
  oneMethod() { }  
  twoMethod() { }  
}
```


```
interface NewInterface {  
  oneMethod()  
  twoMethod()  
}
```

```
class MyClass implements NewInterface {  
  oneMethod() { }  
  twoMethod() { }  
}
```

SCALED AGILE® © Scaled Agile, Inc.

41

Notes:



Activity: Refactoring

Prepare
5 min

- ▶ Step 1: In your workbook, locate the Primes from Knuth with Java translation by Robert Martin
- ▶ Step 2: Look for opportunities to refactor
 - **Hint:** name changes, extract method
- ▶ Step 3: Mark areas for refactoring

SCALED AGILE® © Scaled Agile, Inc. 42

Notes:

Primes from Knuth


```

20 public class GeneratePrimes {
21     /**
22      * @param maxValue
23      *      is the generation limit.
24      */
25     public static int[] generatePrimes(int maxValue) {
26         if (maxValue >= 2) // the only valid case
27         {
28             // declarations
29             int s = maxValue + 1; // size of array
30             boolean[] f = new boolean[s];
31             int i;
32             // initialize array to true.
33             for (i = 0; i < s; i++)
34                 f[i] = true;
35             // get rid of known non-primes
36             f[0] = f[1] = false;
37             // sieve
38             int j;
39             for (i = 2; i < Math.sqrt(s) + 1; i++) {
40                 if (f[i]) // if i is uncrossed, cross its multiples.
41                 {
42                     for (j = 2 * i; j < s; j += i)
43                         f[j] = false; // multiple is not prime
44                 }
45             }
46             // how many primes are there?
47             int count = 0;
48             for (i = 0; i < s; i++) {
49                 if (f[i])
50                     count++; // bump count.
51             }
52             int[] primes = new int[count];
53             // move the primes into the result
54             for (i = 0, j = 0; i < s; i++) {
55                 if (f[i]) // if prime
56                     primes[j++] = i;
57             }
58             return primes; // return the primes
59         } else
60             // maxValue < 2
61             return new int[0]; // return null array if bad input.
62     }
63 }

```




Notes:




Activity: Emergent design

Shout
3 min



Emergent design is...

What is emergent design?



44

Notes:

Emergent design

- ▶ Design emerges as new requirements are added
 - Do not create capabilities that are not needed for current requirements
- ▶ There are tradeoffs
 - Design specifically for current requirement
 - Design generically for next requirement
- ▶ Refactor to add new requirements
 - High-quality code easier to refactor for design changes



SCALED AGILE® © Scaled Agile, Inc.

45

Notes:

Example of changing requirements

Current



GPS/Mapper

- ▶ Speed limit from GPS/Mapper
 - GPS gives current location
 - Mapper retrieves speed limit for that location

Want to add



Speed Sign Identifier

- ▶ Speed limit from signs
 - Camera monitors routes
 - Image processor identifies signs
 - Speed sign processor determines speed limit

SCALED AGILE® © Scaled Agile, Inc.

46

Notes:

Emergent design: Current code (1 of 4)

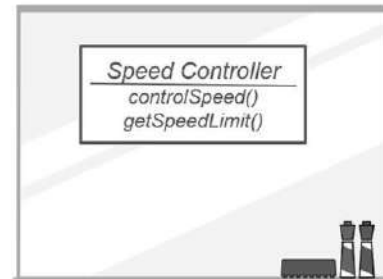
Speed Controller should not be responsible for knowing speed limit

```

class SpeedController {
  void controlSpeed() {
    SpeedLimit desired = getSpeedLimit();
    if (current > desired)
      slowdown();
    else if (current < desired)
      speedup();
  }

  SpeedLimit getSpeedLimit() {
    return GPSTMapper.getSpeedLimit()
  }
}
  
```

</>



SCALED AGILE® © Scaled Agile, Inc.

47

Notes:

Emergent design (2 of 4)

Refactor code – Move Method

```

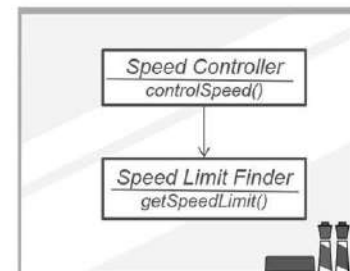
class SpeedController {
  void controlSpeed(){
    SpeedLimit desired =
      SpeedLimitFinder.getSpeedLimit()
    if (current > desired)
      slowdown();
    else if (current < desired)
      speedup();
  }
}
  
```

</>

```

class SpeedLimitFinder {
  SpeedLimit getSpeedLimit() {
    return GPSTMapper.getSpeedLimit()
  }
}
  
```

</>



SCALED AGILE® © Scaled Agile, Inc.

48

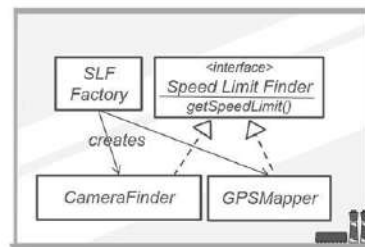
Notes:

Emergent design (3 of 4)

Extract interface, add factory, strategy pattern

```
interface SpeedLimitFinder {
    SpeedLimit getSpeedLimit();
}
```

```
class SpeedLimitFinderFactory {
    SpeedLimitFinder getSLF() {
        switch(configurationVariable) {
            case GPSMapper:
                return new GPSMapper();
            case CameraFinder:
                return new CameraFinder();
        }
    }
}
```



```
class GPSMapper
    implements SpeedLimitFinder {
        SpeedLimit getSpeedLimit(){}
    }
```

```
class CameraFinder
    implements SpeedLimitFinder {
        SpeedLimit getSpeedLimit(){}
    }
```

SCALED AGILE © Scaled Agile, Inc.

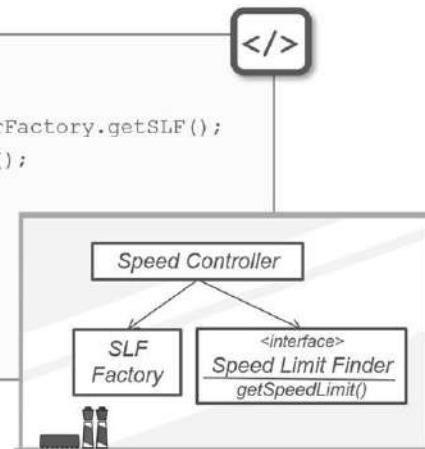
49

Notes:

Emergent design (4 of 4)

Code with factory and interface


```
class SpeedController {
    void controlSpeed() {
        SpeedLimitFinder slf = SpeedLimitFinderFactory.getSLF();
        Speedlimit current = slf.getSpeedLimit();
        if (current > desired)
            slowdown();
        else if (current < desired)
            speedup()
    }
}
```



SCALED AGILE © Scaled Agile, Inc.

50

Notes:




Activity: Build your Agile Software Engineer Plan

Share
2 min

- ▶ **Step 1:** Considering what you have learned, write down at least one improvement item in your *Agile Software Engineer Plan*
- ▶ **Step 2:** Share the item you wrote with the class

In this lesson, we:

- ▶ Designed with tests
- ▶ Applied test-driven development (TDD) practices
- ▶ Implemented test doubles and test data
- ▶ Refactored to support new behavior of the code
- ▶ Practiced emergent design



SCALED AGILE® © Scaled Agile, Inc.

51

Notes:



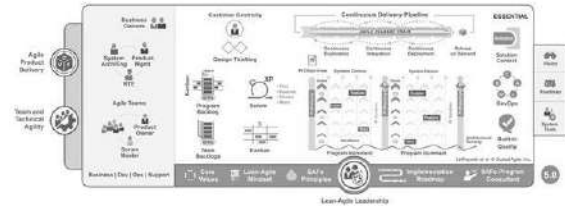
Discussion: Implementing with quality in SAFe

Discuss



► Discuss:

- Who in SAFe is responsible for implementing quality?
- What role does the Architect play in implementing quality?
- Is the PO concerned about implementing with quality?
- When do we do these types of refactoring efforts?



SCALED AGILE® © Scaled Agile, Inc.

52

Notes:



Agile Software Engineering Action Plan

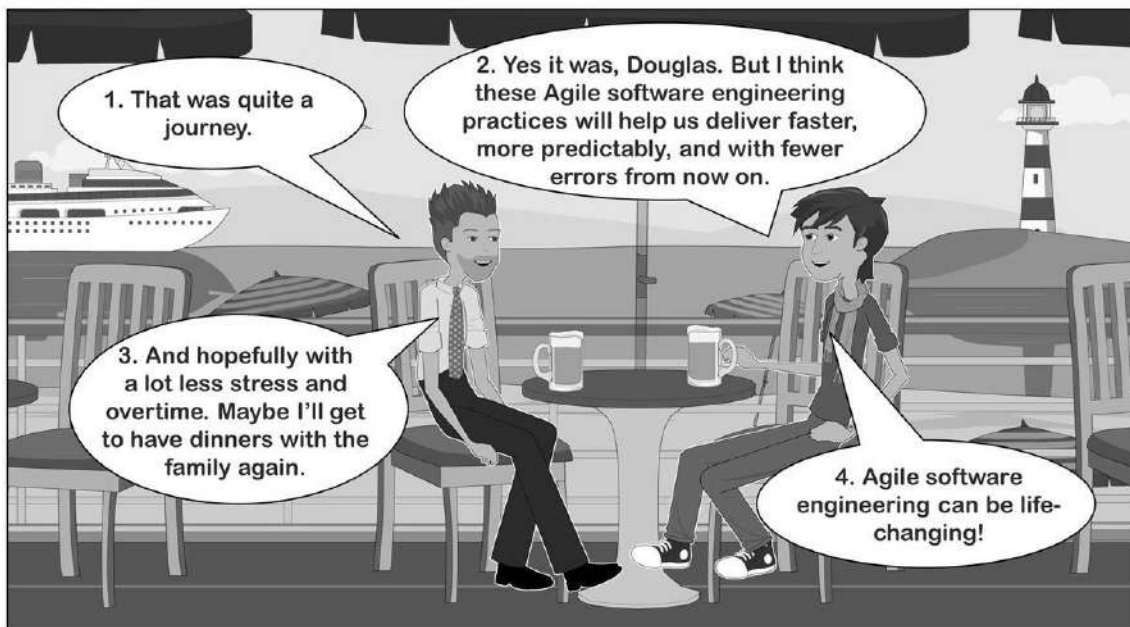
Prepare



- ▶ **Step 1:** Locate the Agile Software Engineer Action Plan at the end of your workbook
- ▶ **Step 2:** Considering what we have covered, write down at least one improvement item in your Agile Software Engineer Action Plan
- ▶ **Step 3:** Share the item you wrote with the class.



Notes:



Notes:

Lesson Review

In this lesson, you:

- 11.1 Design with tests
- 11.2 Apply test-driven development (TDD) practices
- 11.3 Implement test doubles and test data
- 11.4 Refactor to support new behavior of the code
- 11.5 Practice emergent design

Notes:

Lesson 12

Course Review

Learning Objectives:

12.1 Summarize Agile software engineering

12.2 Review your action plan for adopting ASE principles and practices



SAFe® Authorized Course Attending this course gives students access to the SAFe® Agile Software Engineer exam and related preparation materials.

12.1 Summarize Agile software engineering

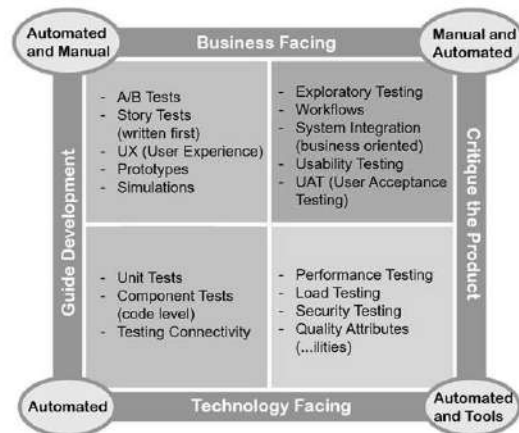
SCALED AGILE © Scaled Agile, Inc.

3

Notes:

Agile software engineering is test-driven

- ▶ Define tests across many dimensions and levels
- ▶ Strive for automation

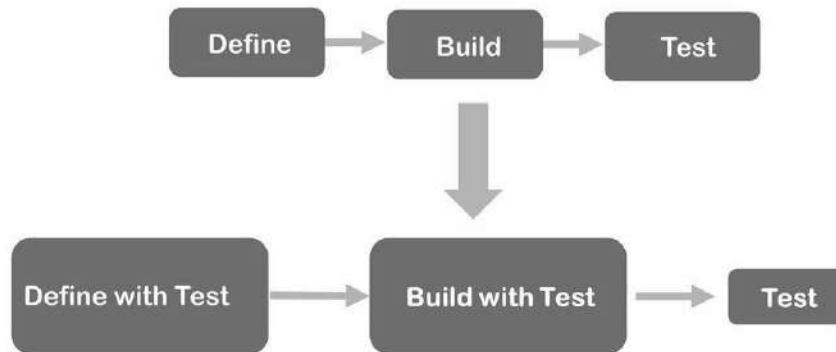


SCALED AGILE © Scaled Agile, Inc.

4

Notes:

Redefining Define-Build-Test

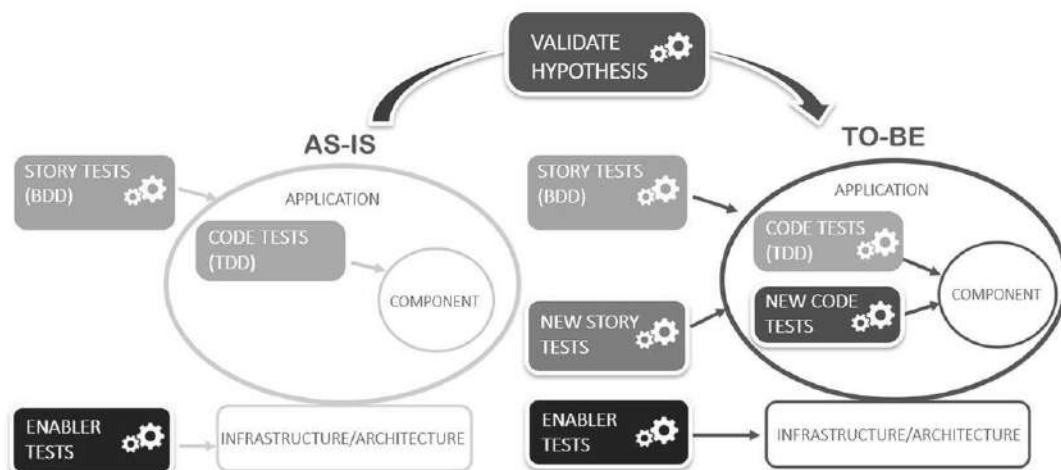


SCALED AGILE® © Scaled Agile, Inc.

5

Notes:

Different types of driven development and tests

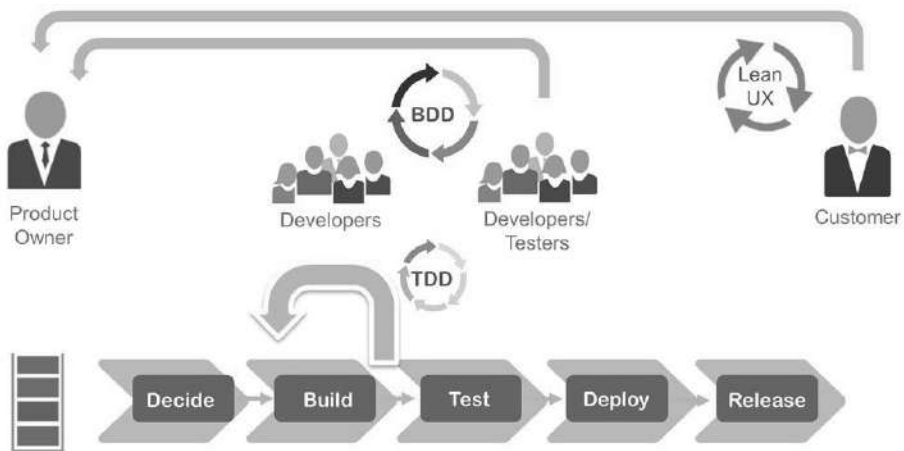


SCALED AGILE® © Scaled Agile, Inc.

6

Notes:

Value flows with multiple feedback loops

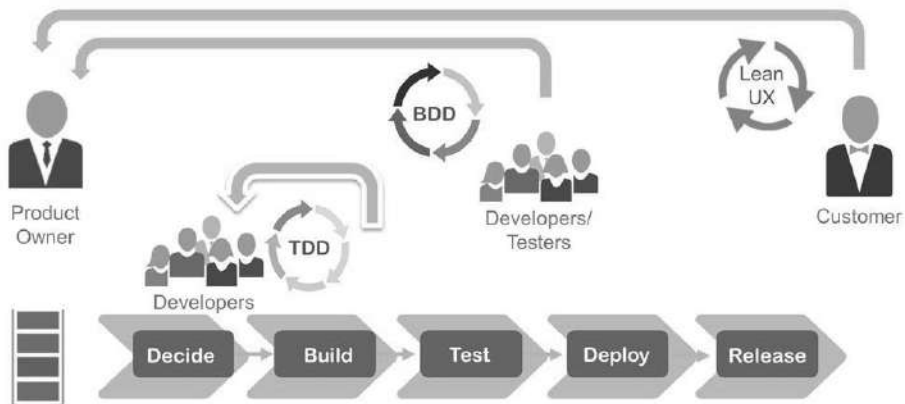


SCALED AGILESM © Scaled Agile, Inc.

7

Notes:

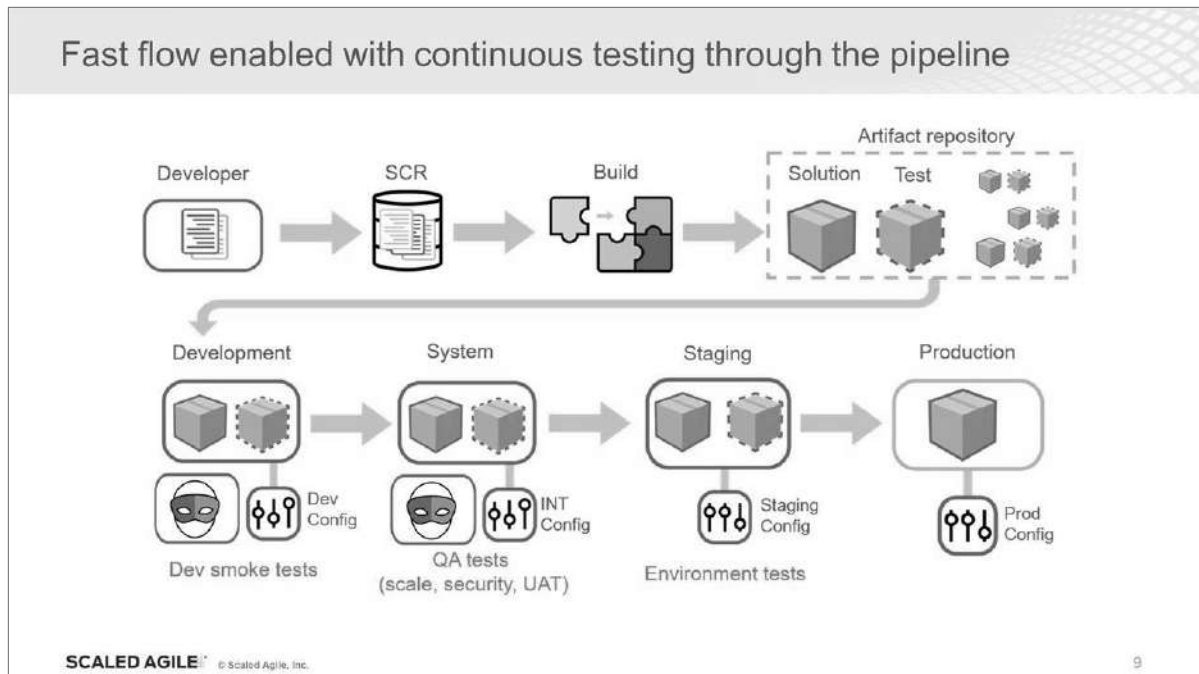
Value flows with multiple feedback loops




SCALED AGILESM © Scaled Agile, Inc.

8

Notes:



Notes:

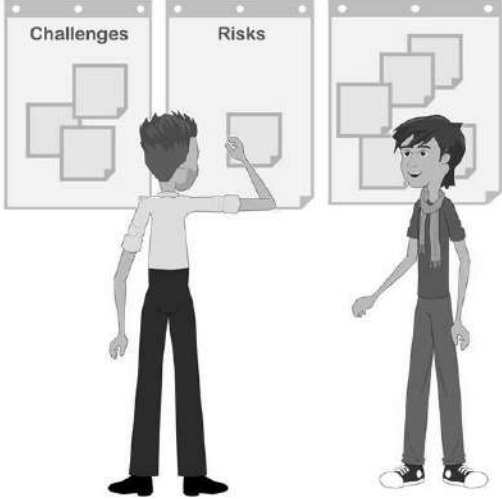


Discussion: Revisiting challenges and risks

Prepare
10 min

Share
3 min

- **Step 1:** Revisit the challenges and risks that you identified in the beginning of the course. Move the challenges or risks you now know how to address to a separate area.
- **Step 2:** As a class, discuss those challenges and risks that you can now (and will) address.




SCALED AGILE® © Scaled Agile, Inc.

10

Notes:



Notes:

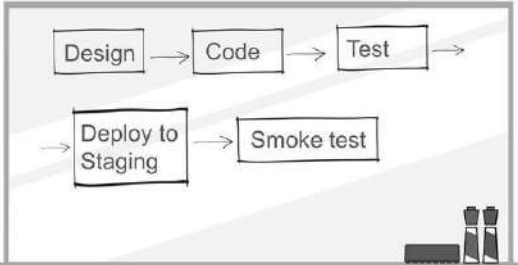


Activity: Improving your Agile software engineering process

Prepare
5 min

Share
5 min

- **Step 1:** Review your current development process from the activity at the beginning of the course, *Diagram current flow for a Feature*
- **Step 2:** Make a list of changes you would like to make to your development process



SCALED AGILE® © Scaled Agile, Inc.

12

Notes:



Activity: Build your Agile Software Engineer Plan



► **Step 1:** You have been crafting your own Agile Software Engineering Plan throughout the course. Review your plan.

► **Step 2:** Think about and share with the class:

- Are the items on your plan realistic and achievable?
- What is the first item you will take action on?
- What your biggest insight from this course?

SCALED AGILE® © Scaled Agile, Inc.

13

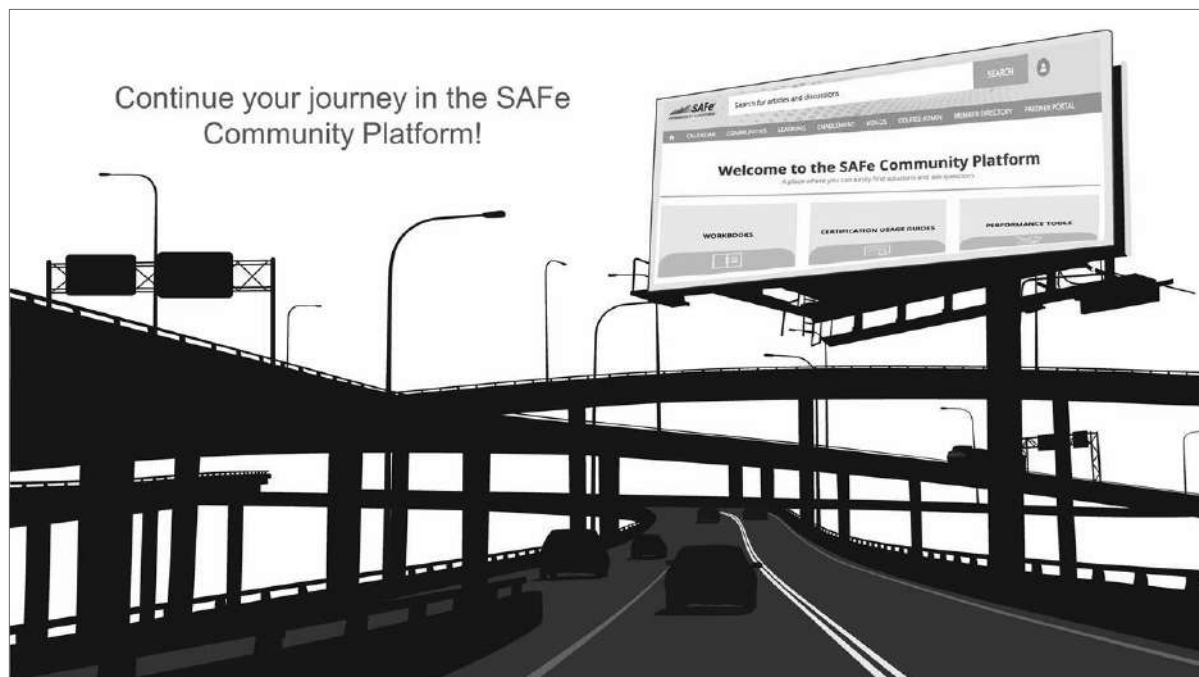
Notes:

Course review

Throughout the course, you:

- ▶ Defined Agile software engineering and its underlying values, principles, and practices
- ▶ Applied the test-first principle to create alignment between tests and requirements
- ▶ Created shared understanding with behavior-driven development (BDD)
- ▶ Communicated with Agile modeling
- ▶ Designed from context for testability
- ▶ Built applications with code and design quality
- ▶ Utilized the test infrastructure for automated testing
- ▶ Collaborated on intentional architecture and emergent design
- ▶ Applied Lean-Agile principles to optimize the flow of value
- ▶ Created an Agile software engineering plan

Notes:



Notes:

Lesson 13

Becoming a Certified SAFe Professional

Learning Objectives:

13.1 Becoming a Certified SAFe Professional



SAFe® Authorized Course Attending this course gives students access to the SAFe® Agile Software Engineer exam and related preparation materials.

Make the most of your learning



Access the SAFe Community Platform

Manage your member profile, continue your learning with toolkits and videos, and access communities of practice and the member directory



Prepare Yourself

Extend your SAFe knowledge and prepare for certification with your learning plan, course workbook, study materials, and practice test before your exam



Become a Certified SAFe Professional

Demonstrate your validated knowledge, skills, and mindset to participate in SAFe methods




Showcase Your SAFe Credentials


Use your digital badge to view global insights, track market labor data, and see where your skills are in demand

SCALED AGILE® © Scaled Agile, Inc.

443

Notes:

**Video: Become a Certified SAFe Professional**


Duration


Continue to build on the foundation of SAFe learning you began in class by studying and taking the certification exam.

Earning this certification demonstrates and establishes your new knowledge.

Certification details at:

<https://www.scaledagile.com/certification/about-safe-certification/>



<https://vimeo.com/307578726>

SCALED AGILE® © Scaled Agile, Inc.

444

Notes:



Video: Welcome to the SAFe Community Platform

Duration
5 min

Want to learn more about the next steps on your SAFe Journey?

Access the SAFe Community Platform and discover all the SAFe resources available for your use!



<https://vimeo.com/201877314>

SCALED AGILE® © Scaled Agile, Inc.

445

Notes:



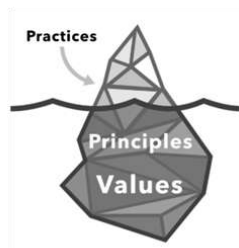
SAFe® for Agile Software Engineer Action Plan

Lesson 1: Enabling Technical Agility for the Lean Enterprise



Considering what you have learned, write down at least one improvement item in your Agile Software Engineer Plan.

Lesson 2: Connecting Principles and Practices to Built-In Quality



Considering what you have learned, write down at least one improvement item in your Agile Software Engineer Plan.



SAFe® for Agile Software Engineer Action Plan

Lesson 3: Accelerating Flow

Considering what you have learned, write down at least one improvement item in your Agile Software Engineer Plan.



Lesson 4: Applying Intentional Architecture

Considering what you have learned, write down at least one improvement item in your Agile Software Engineer Plan.

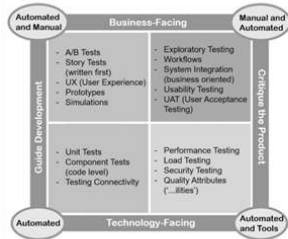


© SCALED AGILE, INC.



SAFe® for Agile Software Engineer Action Plan

Lesson 5: Thinking Test-First



Considering what you have learned, write down at least one improvement item in your Agile Software Engineer Plan.

Lesson 6: Discovering Story Details

- I Independent
- N Negotiable
- V Valuable
- E Estimable
- S Small
- T Testable

Considering what you have learned, write down at least one improvement item in your Agile Software Engineer Plan.



SAFe® for Agile Software Engineer Action Plan

Lesson 7:

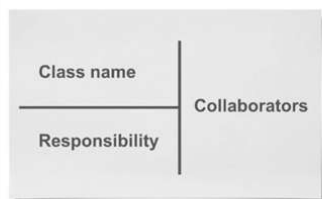
Create a shared Understanding with Behaviour-Driven Development (BDD)



Considering what you have learned, write down at least one improvement item in your Agile Software Engineer Plan.

Lesson 8:

Communicating with Models

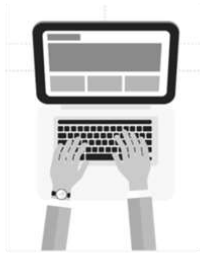


Considering what you have learned, write down at least one improvement item in your Agile Software Engineer Plan.



SAFe® for Agile Software Engineer Action Plan

Lesson 9: Building Systems with Code Quality



Considering what you have learned, write down at least one improvement item in your Agile Software Engineer Plan.

Lesson 10: Building Systems with Design Quality



Considering what you have learned, write down at least one improvement item in your Agile Software Engineer Plan.



SAFe® for Agile Software Engineer Action Plan

Lesson 11: Implementing with Quality



Considering what you have learned, write down at least one improvement item in your Agile Software Engineer Plan.

Glossary



SAFe Glossary:

Visit the Scaled Agile Framework site (<http://v5.scaledagileframework.com/glossary>) to download glossaries translated into other languages