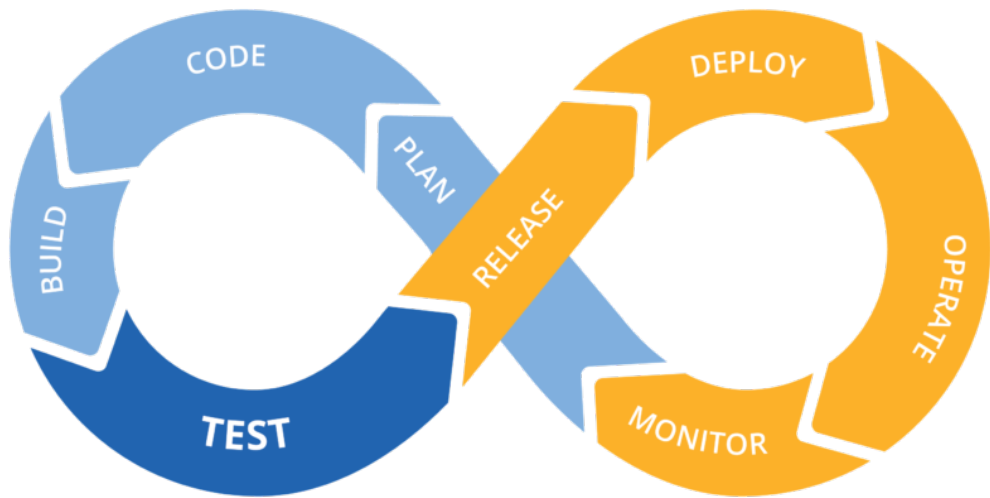


CI/CD Basics & Code Coverage



SWEN-261 Introduction to Software Engineering

Department of Software Engineering
Rochester Institute of Technology

GuessGame

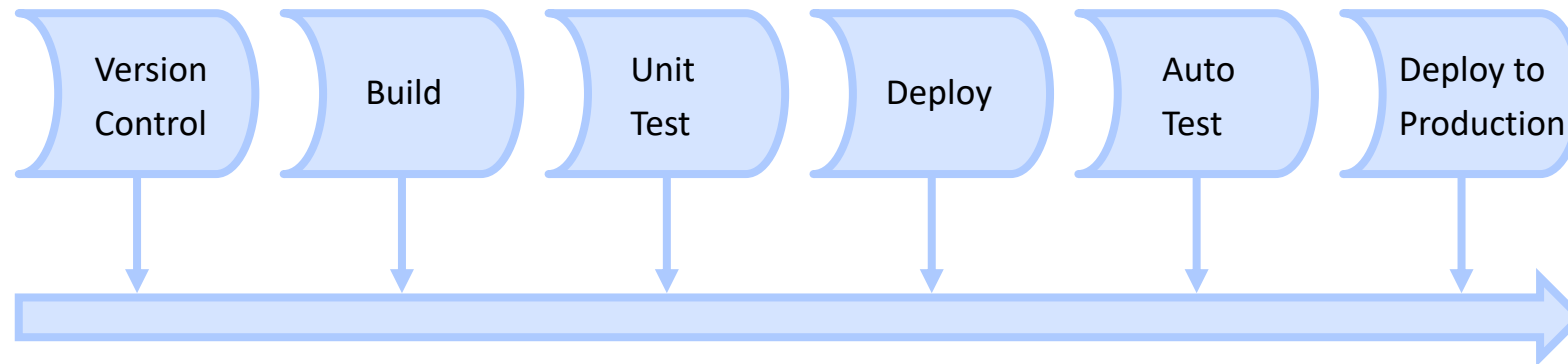
Element	Missed Instructions	Cov.	Missed Branches	Cov.
makeGuess(int)	<div style="width: 93%;"><div style="width: 7%;"></div></div>	93%	<div style="width: 83%;"><div style="width: 17%;"></div></div>	83%
GuessGame(int)	<div style="width: 100%;"></div>	100%	<div style="width: 100%;"></div>	100%
isFinished()	<div style="width: 100%;"></div>	100%	<div style="width: 100%;"></div>	100%
toString()	<div style="width: 100%;"></div>	100%		n/a
isValidGuess(int)	<div style="width: 100%;"></div>	100%	<div style="width: 100%;"></div>	100%
static {...}	<div style="width: 100%;"></div>	100%		n/a
isGameBeginning()	<div style="width: 100%;"></div>	100%	<div style="width: 100%;"></div>	100%
hasMoreGuesses()	<div style="width: 100%;"></div>	100%	<div style="width: 100%;"></div>	100%
GuessGame()	<div style="width: 100%;"></div>	100%		n/a
guessesLeft()	<div style="width: 100%;"></div>	100%		n/a
Total	3 of 148	97%	2 of 28	92%

Model Tier



What is a CI/CD Pipeline?

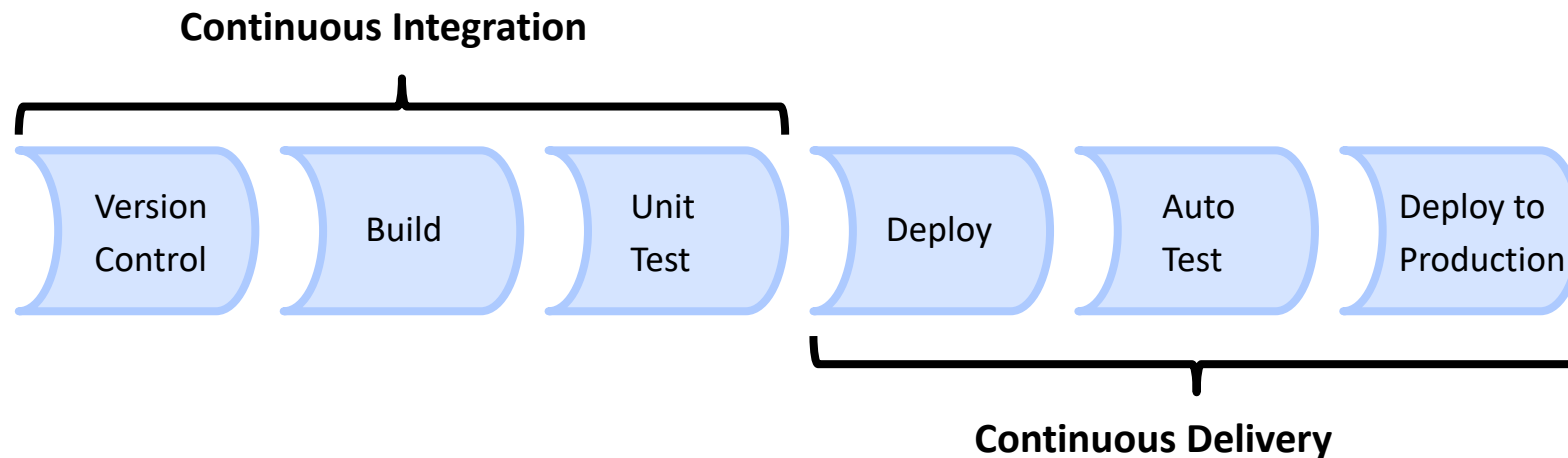
- Continuous Integration/Continuous Delivery (CI/CD) pipelines is a practice focused on improving software delivery using a DevOps approach.
 - *It is a series of steps that must be performed in order to deliver a new version of software.*



- Although it is possible to manually execute each of the steps of a CI/CD pipeline, the true value of CI/CD pipelines is realized through automation

What is a CI/CD Pipeline?

- **Continuous Integration** is development practice where developers integrate all code into a shared repository, frequently.
 - *Once code is merged, the automated build is ready to be verified and tested*



- **Continuous Delivery** executes after Continuous Integration steps have successfully completed
 - *Application is built and deployed to a pre-production environment where additional automated testing and/or Acceptance Testing is conducted before deploying to production*

How good is our Unit Testing?

- We have been running our unit tests but how do we know how well they test all of our code?
 - *Do the unit tests execute across a broad area of code or are they simply testing the same methods and ignoring others?*
- If we are using a CI/CD Pipeline, what if we wanted to impose a certain level of code coverage in our Continuous Integration before we run continue with Continuous Delivery?
- We need a Code Coverage tool!

Code coverage analysis is measuring how well your unit tests exercise the production code.

- Code coverage works like this:
 1. *Compile the project into bytecode*
 2. *Instrument the bytecode with "touch points"*
 3. *Run the unit tests, which gathers coverage data*
 4. *Generate a coverage report from the gathered data*
- There are a few Java coverage tools.
 - *Your project will use JaCoCo*
 - *It integrates well with Maven*
- Having this information is a double-edge sword.
 - *It's mostly a positive thing; telling the team where to spend additional testing effort.*
 - *But don't be obsessed with the metrics; we'll talk more about this later.*

JaCoCo's coverage report is a simple HTML web site that lets you drill down for more information.

- The report is stored in `/target/site/jacoco`.

Top tier

Missed Instructions provides information about the amount of code that has been executed or missed

Missed Branches indicates branch coverage for all if and switch statements

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.heroes.api.heroesapi		88%		n/a	1	4	2	7	1	4	0	2
com.heroes.api.heroesapi.persistence		100%		93%	1	20	0	51	0	12	0	1
com.heroes.api.heroesapi.controller		100%		100%	0	10	0	25	0	8	0	1
com.heroes.api.heroesapi.model		100%		n/a	0	8	0	12	0	8	0	1
Total	5 of 445	98%	1 of 20	95%	2	42	2	95	1	32	0	5

















Controller tier

com.heroes.api.heroesapi.controller				
Element	Missed Instructions	Cov.	Missed Branches	Cov.
HeroController		100%		100%
Total	0 of 110	100%	0 of 4	100%

Class-level

HeroController				
Element	Missed Instructions	Cov.	Missed Branches	Cov.
getHero(int)		100%		100%
deleteHero(int)		100%		100%
searchHeroes(String)		100%		n/a
createHero(Hero)		100%		n/a
updateHero(Hero)		100%		n/a
getHeroes()		100%		n/a
HeroController(HeroDAO)		100%		n/a
static {...}		100%		n/a
Total	0 of 110	100%	0 of 4	100%

It's at the class-level where you can start a meaningful analysis.

Element	Missed Instructions	Cov.	Missed Branches	Cov.
makeGuess(int)		93%		83%
GuessGame(int)		100%		100%
isFinished()		100%		100%
toString()		100%		n/a
isValidGuess(int)		100%		100%
static {...}		100%		n/a
isGameBeginning()		100%		100%
hasMoreGuesses()		100%		100%
GuessGame()		100%		n/a
guessesLeft()		100%		n/a
Total	3 of 148	97%	2 of 28	92%

Model Tier

```
113.     public synchronized GuessResult makeGuess(final int myGuess) {
114.         final GuessResult thisResult;
115.         // validate arguments
116.         if (myGuess < 0 || myGuess >= UPPER_BOUND) {
117.             thisResult = GuessResult.INVALID;
118.         } else {
119.             // assert that the game isn't over
120.             if (howManyGuessesLeft == 0) {
121.                 throw new IllegalStateException("No more guesses allowed.");
122.             }

```

Color Legend

Green → covered
Yellow → partially covered
Red → not covered

- This `GuessGame` code had 97% coverage. So, what do you do?
- On the one hand:
 - *That's REALLY good already.*
 - *The only missing test is a defensive check so maybe say "that's good enough."*
- On the other hand:
 - *This is a core Model tier class.*
 - *We want these to be "friendly" test dependencies.*
 - *So maybe the team agrees to make this 100% covered.*
- What tests need to be added?

There needs to be a test to check making an invalid guess.

- Here's a test:

```
@Test
public void make_an_invalid_guess() {
    final GuessGame CuT = new GuessGame();
    assertEquals(CuT.makeGuess(TOO_SMALL), GuessResult.INVALID);
    assertFalse(CuT.isFinished(), "Game is not finished");
}
```

- Here's the updated analysis:

```
113.     public synchronized GuessResult makeGuess(final int myGuess) {
114.         final GuessResult thisResult;
115.         // validate arguments
116.         ◆ if (myGuess < 0 || myGuess >= UPPER_BOUND) {
117.             thisResult = GuessResult.INVALID;
118.         } else
```

This line is tested but only through this part of the branch.

We need to test the second part of the branch.

If we test that second branch, we should be there.

- Here's a test of a guess that is too big:

@Test




```
public void make_an_invalid_guess_too_big() {  
    final GuessGame CuT = new GuessGame();  
    assertEquals(CuT.makeGuess(TOO_BIG), GuessResult.INVALID);  
    assertFalse(CuT.isFinished(), "Game is not finished");  
}
```

- Here's the updated analysis:

```
113.     public synchronized GuessResult makeGuess(final int myGuess) {  
114.         final GuessResult thisResult;  
115.         // validate arguments  
116.         if (myGuess < 0 || myGuess >= UPPER_BOUND) {  
117.             thisResult = GuessResult.INVALID;  
118.         } else {
```

- Now the Model tier is fully tested!

com.example.model

Element	Missed Instructions	Cov.	Missed Branches	Cov.
GuessGame		100%		100%
GuessGame.GuessResult		100%		n/a
Total	0 of 197	100%	0 of 28	100%

Model Tier

Deciding what level of coverage depends upon several factors...

- Some components (Model tier) are used across multiple other architectural tiers.
 - *We recommend 95% or better for Model tier.*
- Others are only used by the REST API.
 - *We recommend 90% or better in all other tiers.*
- Other factors:
 - *Team and company culture*
 - *Application domain*
 - ◆ Regulatory requirements may specify testing requirements.
 - ◆ Those defensive checks may be safety checks. You can not know if the system is safe if you do not test the checks.



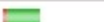

The coverage data is cumulative across all tests which may make results look better than they are.

- You want to gather coverage data from unit tests of a class not use of the class by tests of other classes.
 - *The ultimate is to test one class at a time which is not reasonable.*
 - *A reasonable compromise is measure code coverage for testing one tier at a time.*
- The JUnit framework and build tools allow that.
 - `@Tag("name")` *each test file to place it into a tier category. Use Model-tier, Controller-tier, etc.*
 - *Reset the coverage data after each tier is tested and generate the report in a separate location.*

```
13 @Tag("Model-tier")
14 public class HeroTest {
15     @Test
16     public void testDefaultCtor() {
17         // Setup
18         int expected_id = Hero.DEFAULT_ID;
19         String expected_name = Hero.DEFAULT_NAME;
20
21         // Invoke
22         Hero hero = new Hero();
23
24         // Analyze
25         assertEquals(expected_id, hero.getId());
26         assertEquals(expected_name, hero.getName());
27     }
```

Your project's pom.xml file has several test execution ids defined.

- Clean the target directory, and run all three tier-based tests
 - `mvn clean test`
 - **The reports are in `/target/site/jacoco/tier/index.html` where `tier` contains the coverage report for that specific tier**
- We can also run a check to ensure our coverage has not dropped
 - `mvn clean verify`

Element	Missed Instructions	Cov.
<code>com.heroes.api.heroesapi.controller</code>		25%
<code>com.heroes.api.heroesapi.model</code>		67%
<code>com.heroes.api.heroesapi</code>		88%
<code>com.heroes.api.heroesapi.persistence</code>		100%
Total	103 of 445	76%

Code coverage target set to 90%, yet we are only at 76%

```
[WARNING] Rule violated for bundle heroes-api: instructions covered ratio is 0.76, but expected minimum is 0.90
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
```