Software Application Engineering

Introduction to tutorials

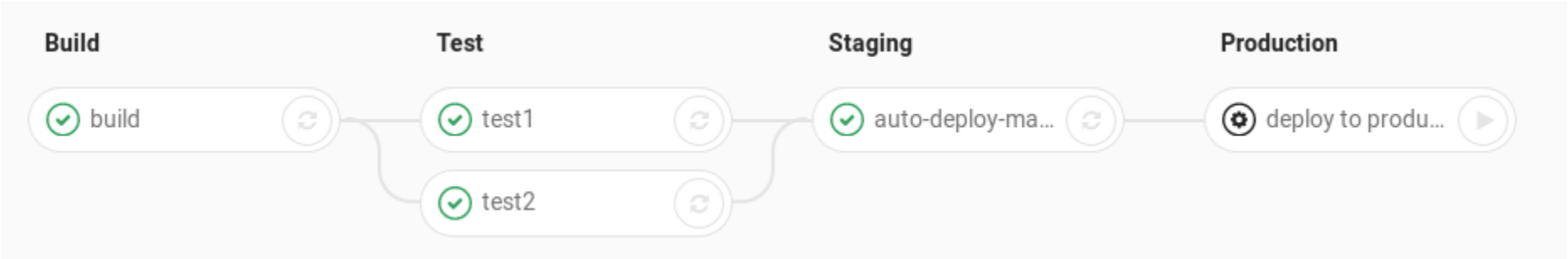
- CI/CD with GitLab
- JUnit
- SonarQube

CI/CD with GitLab

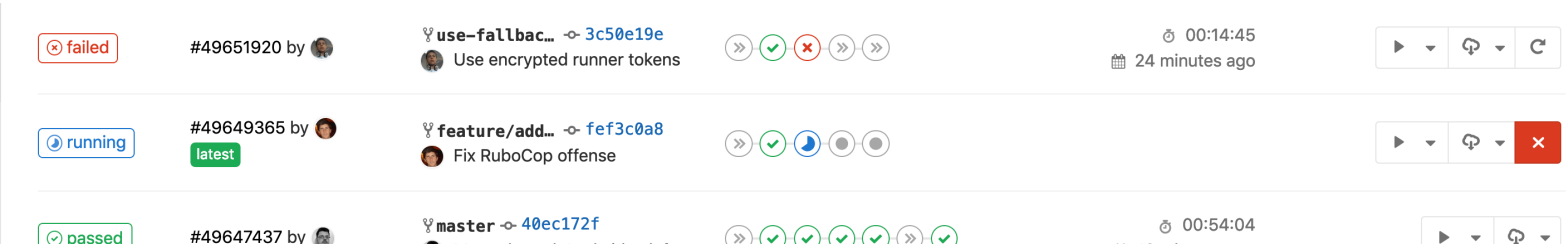
- Code base: “hangman” application, made of 4 Java classes, dependencies and main class declared in a pom.xml file
- Objective: each time a new version of any of these 5 files is pushed to GitLab, automatically:
  - Run unit tests, assess test coverage and publish reports
  - Build an executable jar and publish it
  - Build a docker image embedding the executable jar and publish it
  - Monitor the evolution of code quality (such as: number of “bad code smells”, percentage of comments…)

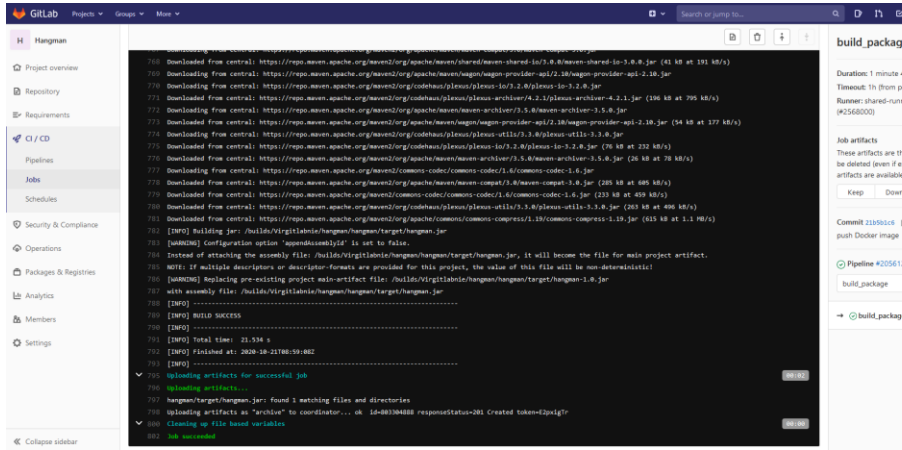
GitLab CI/CD Principles

- ".gitlab-ci.yml" file at the root of the project
- Declares a ***pipeline*** = set of sequential ***stages***, and ***jobs*** to be executed concurrently during each stage



- GitLab Runners then execute the pipeline and display the logs in an online terminal
- GitLab displays the status of the pipeline





Tuto #1: from source code to Docker image

- Repository
  - Files
  - Commits
  - Branches
  - Tags
  - Contributors
  - Graph
  - Compare
  - Locked Files
- Requirements
- CI / CD
- Security & Compliance
- Operations
- Packages & Registries
- Analytics

✔ This GitLab CI configuration is valid. Learn more

.gitlab-ci.yml 644 Bytes

```
stages:
    - build_package
    - create_docker_image

build_package_job:
    stage: build_package
    image: maven:latest
    script:
    - mvn -f hangman/pom.xml clean package assembly:single
    artifacts:
        paths:
            - hangman/target/hangman.jar
        expire_in: 15 mos 2 days 3 hrs 4 mins 5 sec

services:
  - docker:dind
create_docker_image_job:
    stage: create_docker_image
    image: docker:latest
    script:
        - docker build -t $CI_REGISTRY_IMAGE/hangman -f Dockerfile.hangman .
        - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
        - docker push $CI_REGISTRY_IMAGE/hangman
```

Tuto #2: unit tests and test coverage

- Junit: framework for unit testing Java code
- Call a method with specified values and check properties on the result (not null, equals to an expected value, ….)
- Example:

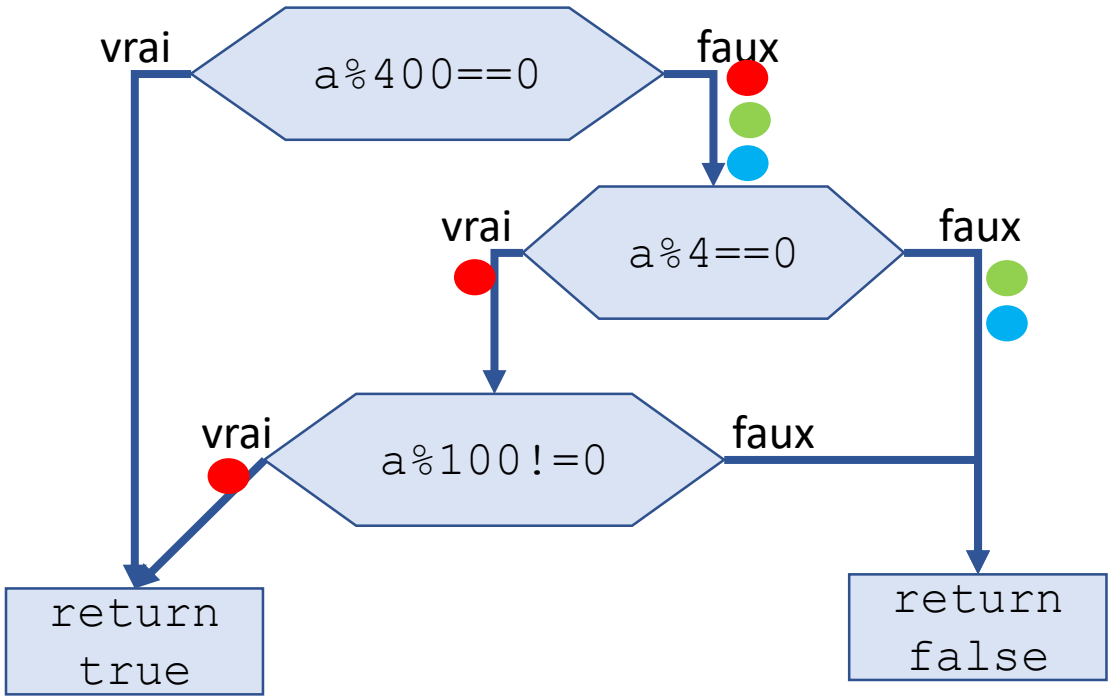
```
public class Parite {
    public boolean isPair(int number){
       return(number%2 == 0){
    }
}
```

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class PariteTest {
    @Test
    public void testIsPair(){
        Parite p = new Parite();
          assertEquals(true, p.isPair(10));
    }
}
```

example: java2novice.com

Test coverage

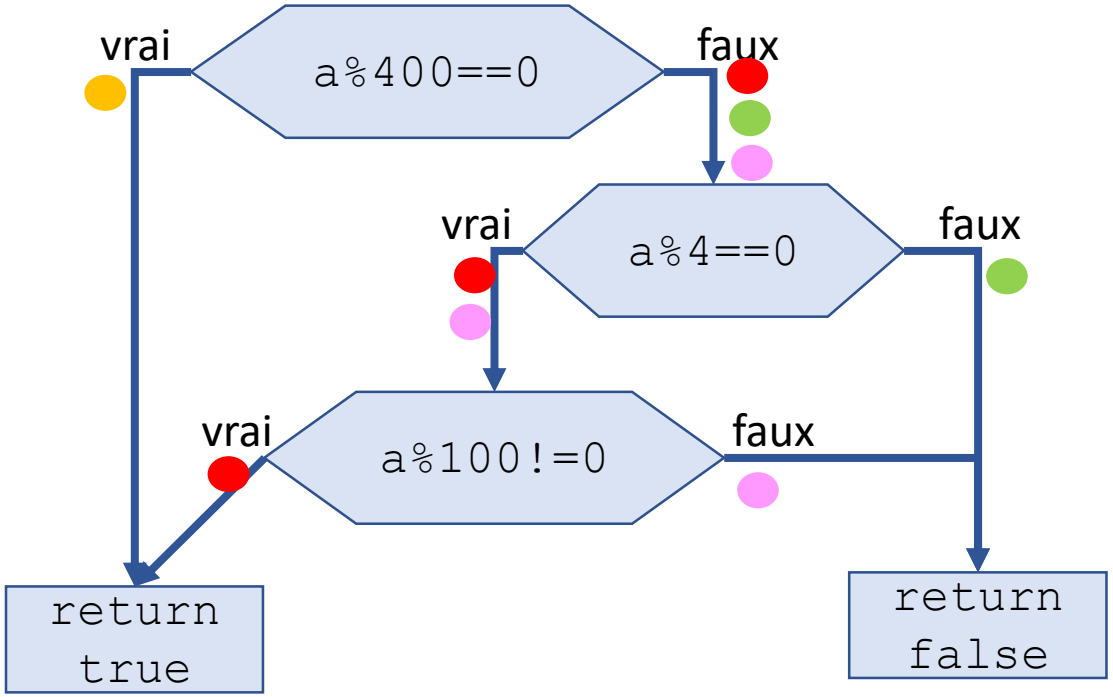
```
public static boolean isBissextile(int annee) {
    return (annee % 400 == 0) || ((annee % 4 == 0) && (annee % 100 != 0));
}
```



2016
2017
2018

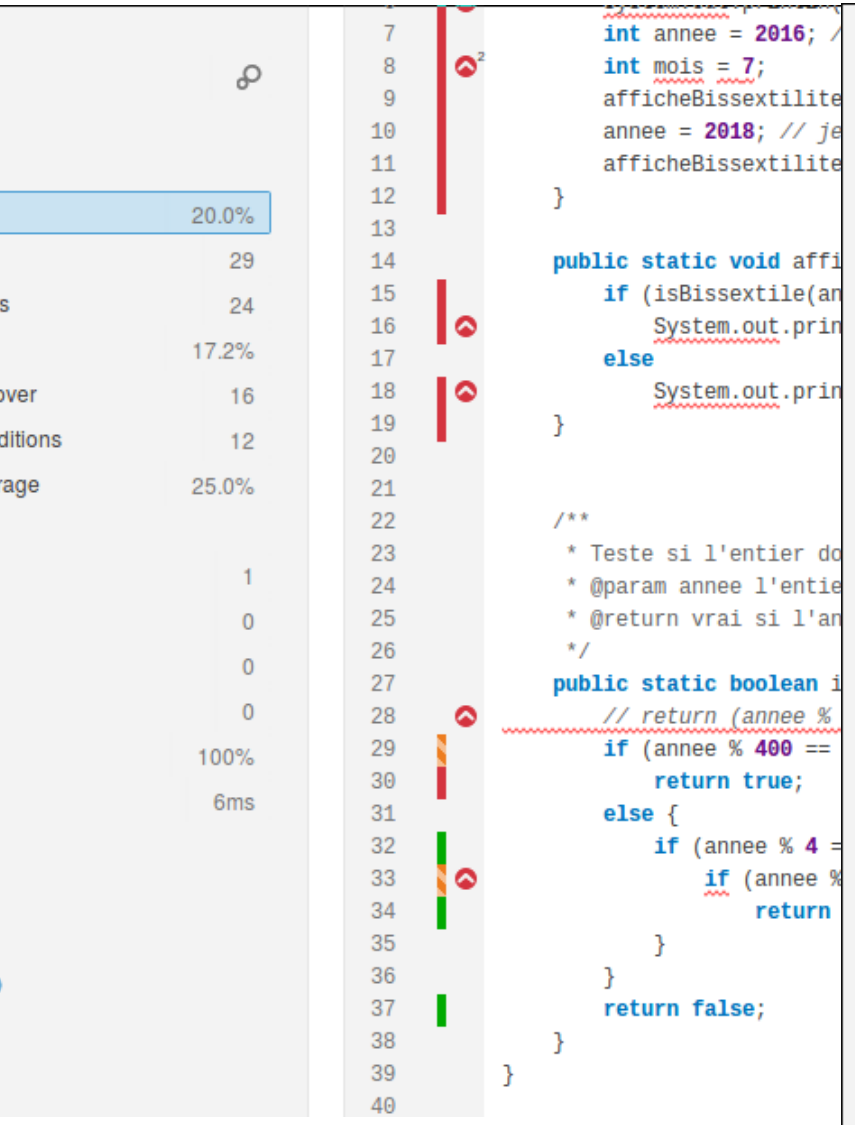
Better test coverage


```java
public static boolean isBissextile(int annee) {
    return (annee % 400 == 0) || ((annee % 4 == 0) && (annee % 100 != 0));
}
```

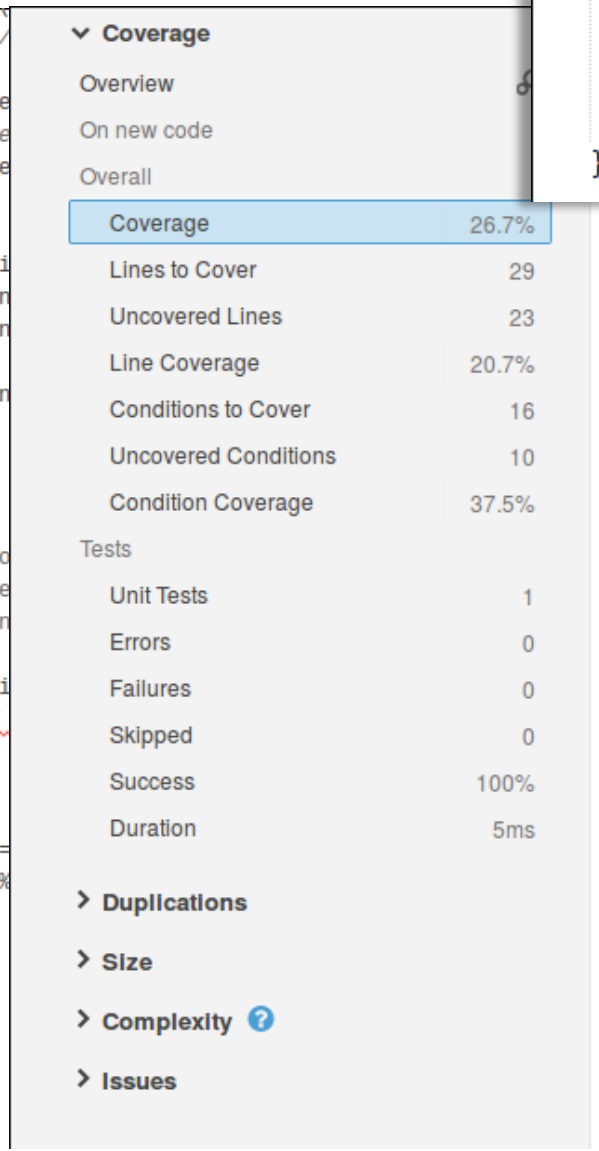


- 2016
- 2017
- 2000
- 1900

Test coverage visualization




```
7   int annee = 2016; /
8   int mois = 7;
9   afficheBissextilite
10  annee = 2018; // je
11  afficheBissextilite
12  }
13
14  public static void affi
15      if (isBissextile(an
16          System.out.prin
17      else
18          System.out.prin
19  }
20
21
22  /**
23   * Teste si l'entier do
24   * @param annee l'entie
25   * @return vrai si l'an
26   */
27  public static boolean i
28      // return (annee %
29      if (annee % 400 ==
30          return true;
31      else {
32          if (annee % 4 =
33              if (annee %
34                  return
35          }
36      }
37      return false;
38  }
39  }
40
```




```
@Test
public void testIsBissextile() {
    int annee = 2016;
    boolean expResult = true;
    boolean result = Conventions.isBissextile(annee);
    assertEquals(expResult, result);

    assertEquals(false, Conventions.isBissextile(2017));
    assertEquals(true, Conventions.isBissextile(2000));
    assertEquals(false, Conventions.isBissextile(1900));
}
```

```
16          System.out.println(annee + " est une année bissextile");
17      else
18          System.out.println(annee + " n'est pas une année bissextile"
19  }
20
21
22  /**
23   * Teste si l'entier donné correspond à une année bissextile ou non
24   * @param annee l'entier représentant l'année à tester
25   * @return vrai si l'année est bissextile, faux sinon
26   */
27  public static boolean isBissextile(int annee) {
28      // return (annee % 400 == 0) || ((annee % 4 == 0) && (annee % 10
29      if (annee % 400 == 0)
30          return true;
31      else {
32          if (annee % 4 == 0) {
33              if (annee % 100 != 0)
34                  return true;
35          }
36      }
37      return false;
38  }
39  }
40
```

Tuto #3: Code quality monitoring with SonarQube

- Identify
  - Bad code smells
  - Code duplication
  - [Test coverage]
  - Security hot-spots
  - …
- Track the evolution of metrics