SURVEY SUMMARY

- Why didn't we have this online: Typically more feedback received when survey done in class, rather than online.
- Level of challenge: mixed feelings
  - on average happy with the level of challenge;
  - some would prefer slightly less challenging course – more support from us (check updated website) + better time management from you.
  - some would prefer a more challenging course – action point for me: give pointers to advanced resources.

SURVEY SUMMARY

Things that work well:

- Weekly tests (++);
- Lectures/tips on different aspects of CSLP/ parts not understood by most people;
- Support on Piazza.

SURVEY SUMMARY

Improvements (I):

- More info on testing – a short discussion today;
- More testing + info about schedule
    - Every Sunday;
    - Additional testing before deadlines – Part 2 on Wed;
    - Future: token based system.
- Creating simulation states – quite specific, happy to chat during office hours;
- Weekly lab: something for the DoT;

SURVEY SUMMARY

Improvements (II):

- More clarity on valid/invalid input and what is being tested;
  - I gave an overview of what we are testing for Part 2 during last lecture;
  - Precise instructions would remove much of the personal contribution and diminish learning outcomes;
  - In the future we may consider a set of known tests (e.g. 70%) and completely hidden ones (30%).

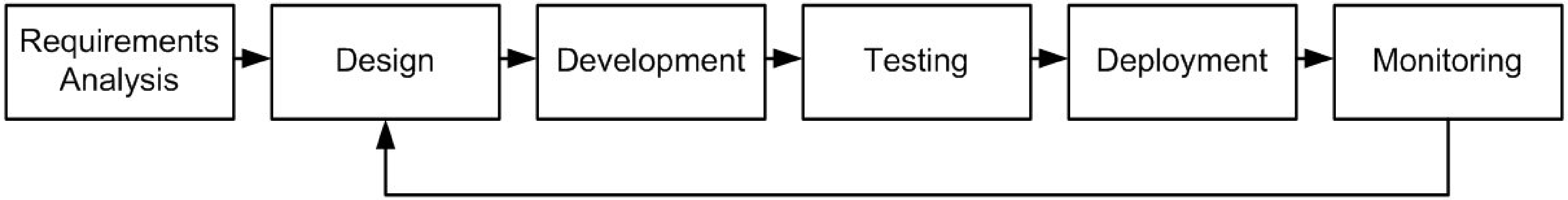
SURVEY SUMMARY

Too much (?):

- Software engineering aspects;
  - Pre-course survey suggested class interest in code optimisation;
  - Some responses asked precisely for more info about testing (which is SE related);

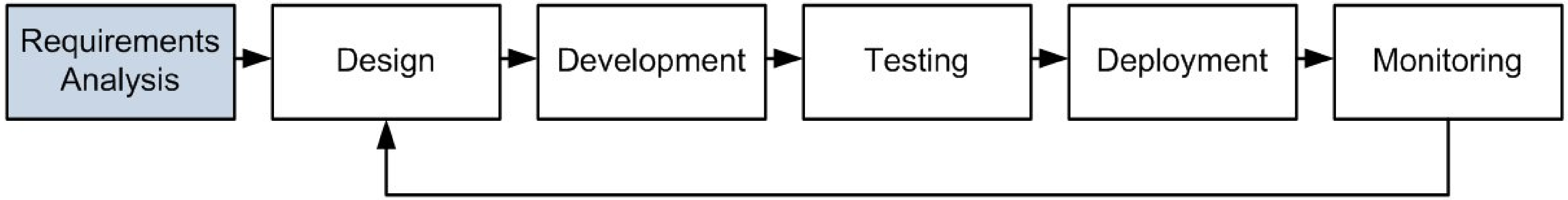
DESIGN ASPECTS

SYSTEM/PROCESS IMPLEMENTATION



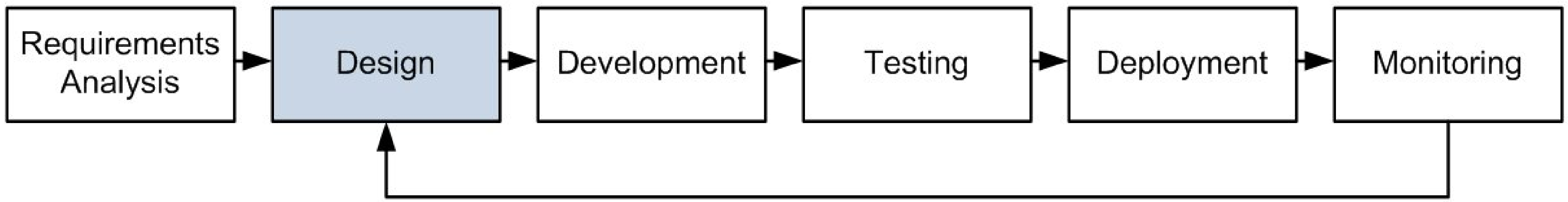
- Designing and implementing logistics operations, complex processes, and systems involves several steps.
- There is often a feedback loop involved, which allows to refine/improve/extend the system.

REQUIREMENTS ANALYSIS



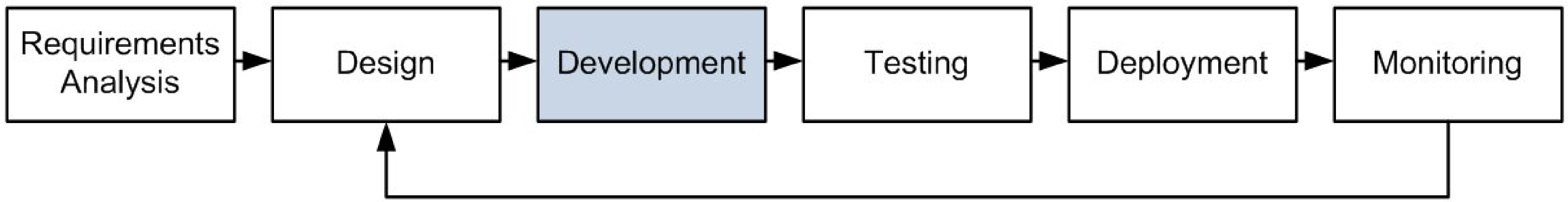
- Understand the problem domain and specifications, and identify the key entities involved.
- Build an abstract representation of the system to be able to handle various input scenarios.

SYSTEM DESIGN



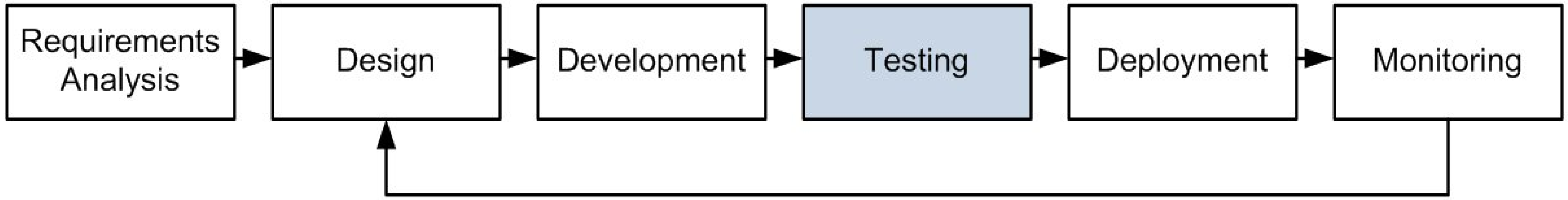
- Divide the system into components; choose suitable methodologies for implementing each component.
- Define appropriate data structures, input/output formats, and so on.

DEVELOPMENT



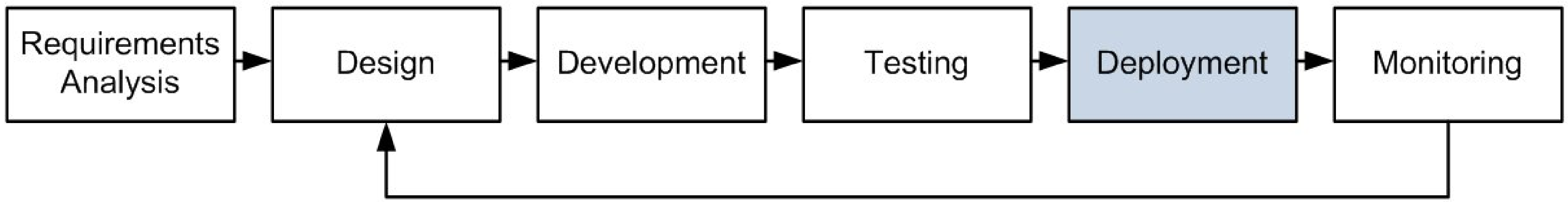
- This is the actual implementation work and is typically coupled with some preliminary testing.
- For source code, janitorial work, refactoring and some optimisations are also performed at this stage.

TESTING



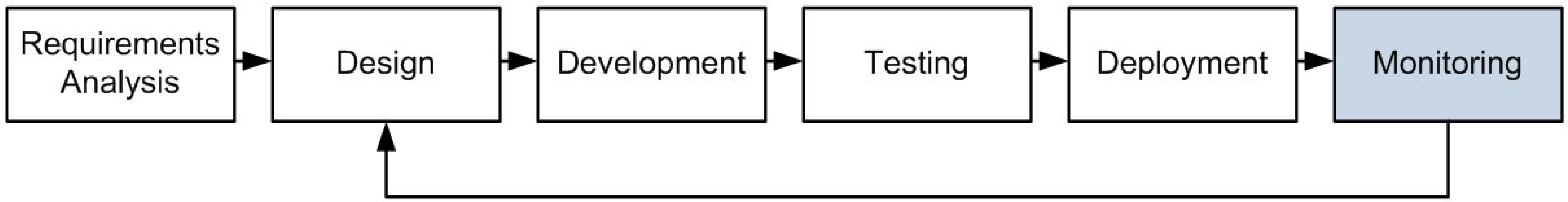
- Validation is performed once the system is partially/ entirely developed; also benchmarking and profiling.
- A system's *performance evaluation* is undertaken (experimentation with different inputs, distributions).

DEPLOYMENT



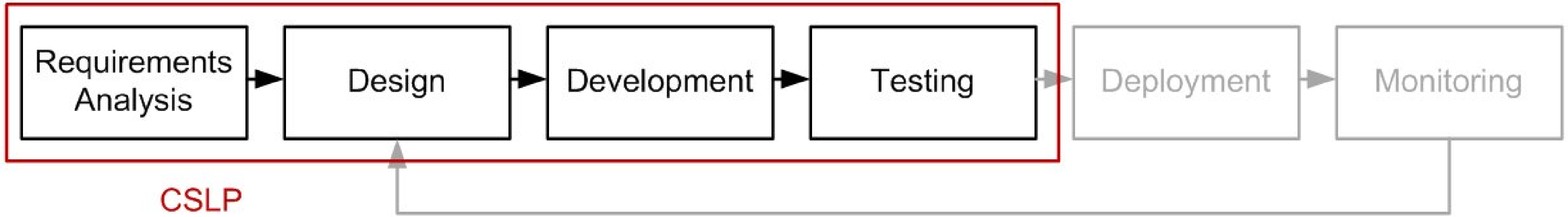
- Once the tool (planner, simulator, etc.) has been thoroughly tested it can be deployed in a real setting.
- The input will be based on actual data and inputs may change over time (e.g. based on certain events).

MONITORING



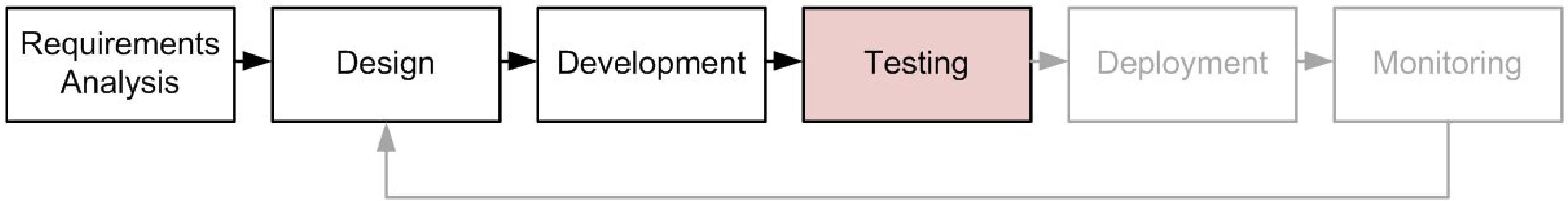
- Once the system is operational, it is possible to gather real measurements and use those to refine the design.
- If new requirements are identified during operation, the system can be further extended.

THE BIN SERVICE PROCESS



- Your simulator will be implementing a good bit of what could become a real logistics system.
- Unfortunately you will not have the opportunity to experiment with real data, but (time permitting) you have the flexibility to develop additional features.

PERFORMANCE EVALUATION



- We have discussed the requirements, as well as different design and development aspects for your simulator.
- We will now look into performance evaluation issues. Some of the things I will present may not be needed for this assignment, but will likely prove useful later.

PERFORMANCE EVALUATION

- Generally speaking, this is about quantifying the performance of a system.
- The first step is to identify the relevant *metrics*, i.e. measurable quantities that capture properties of interest.
  - This could the throughput of a network link, the power consumption of a mobile device, the memory used by a software application, etc.
  - For CSLP we are interested in the average trip duration, number of trips per schedule, trip efficiency, average volume collected, percentage of overflowed bins.

METRICS

- It is essential to understand the performance evaluation goals, i.e. whether a metric should be small or large.
- It is also important to be aware of the goals of the evaluation:
  - Improve the dimensioning/parametrisation of a system or process.
  - Compare how different designs perform under different inputs and chose the best one.

METHODOLOGIES

When designing a system, performance evaluation can be conducted through one or more of the following methodologies:

1. **Numerical analysis** - plugging some numerical values into a mathematical model of the system and computing the metrics of interest.
2. **Simulation** - constructing a simplified model of a more complex real system and simulating its behaviour; typically fast, but neglecting certain practical aspects.
3. **Experimentation** - Analysing the performance of a system via measurements. Assessing performance under exceptional circumstances may be infeasible.

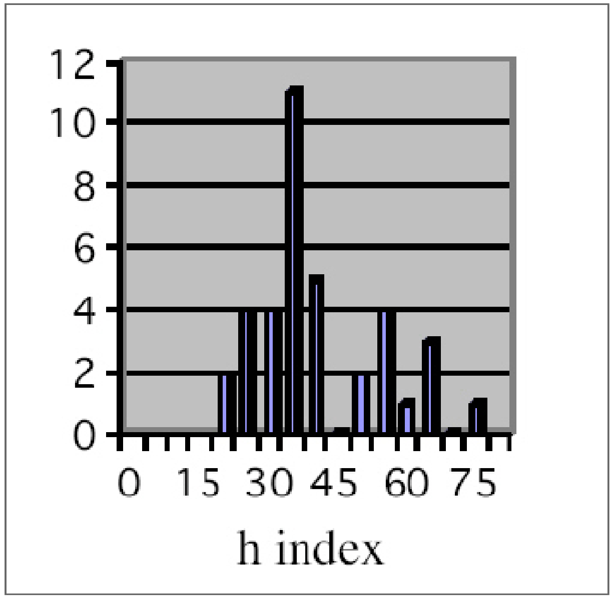
ACCURACY

- It is advisable that the assumptions made for the evaluation campaign are well documented, to ensure the tests performed are *reproducible*.
- You are working with a stochastic simulator and thus there will be some variability in the results of different tests with the same input.
- For this practical you have been asked to give average values of a set of metrics.
- In rigorous studies, it is necessary to also provide some confidence intervals for the results.

SUMMARY STATISTICS

Histograms are graphical representations of the distribution of a set of measurements.

Example: distribution of the *h*-index of Nobel-prize recipients in Physics between 1985-2005.



Source: J.E. Hirsch, "An index to quantify an individual's scientific research output", Proc. NAS, 2005.

h-index: number of papers with *h* or more citations.

HISTOGRAMS

- In mathematical terms, the histogram is a function that counts the number of observations in different categories (bins – not to be confused with waste bins in simulator)
- The number of bins is typically computed as

$$k = \frac{\max(\mathbf{x}) - \min(\mathbf{x})}{\sqrt{n}}$$

where n is the number of samples in the data set x.

MEAN AND STANDARD DEVIATION

Computing the *mean* (average) of a set of measurements is straightforward:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$$

The *standard deviation* gives a measure of the variation of the measurements from the mean:

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2}$$

CONFIDENCE INTERVALS

- These can be used to quantify the uncertainty about the average of a set of measurements subject to randomness.
- When computing averages across multiple simulations, you are gathering samples to estimate an unknown population mean.
- You choose the significance level that will reflect how confident you can be that the true value lies within that interval,
- E.g. for a significance level of 0.05, you will obtain a 95% confidence interval (typically used in practice).

CONFIDENCE INTERVALS

- The width of the confidence is affected by:
  - sample size,
  - population variability (standard deviation),
  - confidence level chosen.
- Central Limit Theorem: For a large sample size, the sampling distribution of the mean will approach a *normal distribution*.
- The sample mean and the mean of the population are identical.

CONFIDENCE INTERVALS

A quick method to compute a CI is:

$$\mu \pm z_{\alpha/2} \frac{\sigma}{\sqrt{n}}$$

where $z_{\alpha/2}$ is the critical coefficient corresponding to a confidence level α; and is obtained from z-score tables.

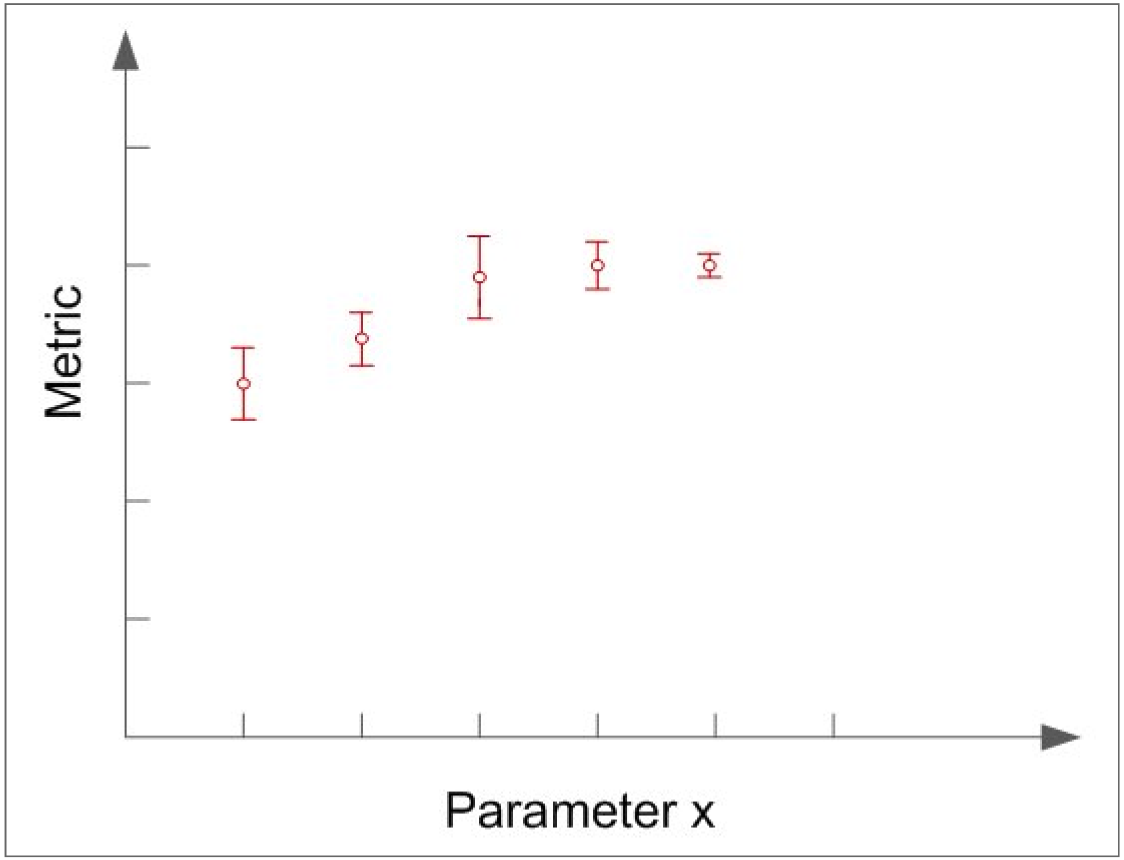
Example: Sample size 20, mean 10, standard deviation 1.45, 95% confidence level, i.e. a critical coefficient corresponding to a z-score of 0.475, which is 1.96.

CI is 20±0.02

i.e. [19.8, 20.2]

CONFIDENCE INTERVALS

Plotting CIs



NOTES ON UNIT TESTING

UNIT TESTING

- This refers to testing different components (units) you develop;
- More common in large collaborative projects where different people are responsible for different (independent) functionality;
- Goal: verify that a particular component works as expected:
  - The output returned makes sense / is correct;
  - Errors are handled appropriately.
- Typically involves writing a tests suite, which one can run at any stage.

UNIT TESTING

- If you later decide to add more functionality, well tested code will allow you to easily identify limitations of the new features.
- You will probably develop separate new tests for the new features.
- The only cost is time: you need time to write these tests, but the quality of the resulting software may be worth it.

WHAT'S INVOLVED

- Many languages offer unit testing frameworks
  – see list **here**.
- Sometimes use stubs / mock objects to isolate a particular unit of work you want to test.
- The idea is to implement some abstract interface that simply allows the component you are testing to work, without actually doing anything.
- Similarly you may also implement drivers, which primarily call the unit you want to test.

A JAVA EXAMPLE

Suppose you want to test this:

```
public class Summator{
  public int computeSum(int[] values) {
    int sum = 0;
    for (int item : values)
      sum += item;
    return sum;
  }
}
```

A JAVA EXAMPLE

Prepare the test

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class TestOne {
  @Test
  public void checkResult() {
    Summator s = new Sumator();
    int[] numbers = {1,2,3,5,7};
    int result = s.computeSum(numbers);
    assertEquals(18, result);
  }
}
```

A JAVA EXAMPLE

Running the test outputs

```
iJUnit version 4.12
.
Time: 0,006

OK (1 test)
```

A JAVA EXAMPLE

Assuming you modify the code of the 'Summator' (e.g. return before computing anything) and then re-run the test:

```
JUnit version 4.12
.E
Time: 0,007
There was 1 failure:
1) checkResult(TestOne)
java.lang.AssertionError: expected:<18> but was:<0>
  at org.junit.Assert.fail(Assert.java:88)
  ...

FAILURES!!!
Tests run: 1,  Failures: 1
```

This tells you the number of failures, where the problems were, and what went wrong.

FINAL REMARKS

- Admittedly you may not need sophisticated unit testing for CSLP, but if you want to learn more, the **JUnit wiki** is a good place to start.
- **PyUnit** gives you the Python version of JUnit. Similar frameworks available for other languages.
- You may even want to take a test driven development (TDD) approach, i.e. write a test; see the code fail; write code to pass the test; repeat.