Test-Driven Development

Joachim Vandekerckhove

Test-driven development

What is test-driven development (TDD)?

TDD is a software development methodology where tests are written for a feature before writing the code for that feature.

What is test-driven development (TDD)?

TDD is a software development methodology where tests are written for a feature before writing the code for that feature.

Yes that sounds weird, but let's give the computer scientists some credit. We actually do it all the time intuitively.

Steps of TDD

1. Write a failing test.

Steps of TDD

1. Write a failing test.

   The test fails because the code it tests doesn't exist yet or it is not implemented yet.

Steps of TDD

1. Write a failing test.

   The test fails because the code it tests doesn't exist yet or it is not implemented yet.

2. Write just enough code to make the test pass.

Steps of TDD

1. Write a failing test.

   The test fails because the code it tests doesn't exist yet or it is not implemented yet.

2. Write just enough code to make the test pass.

   Write the minimum amount of code necessary to pass the test. Don't add any additional features or improvements.

Steps of TDD

1. Write a failing test.

   The test fails because the code it tests doesn't exist yet or it is not implemented yet.
2. Write just enough code to make the test pass.

   Write the minimum amount of code necessary to pass the test. Don't add any additional features or improvements.
3. Refactor the code as needed.

Steps of TDD

1. Write a failing test.

   The test fails because the code it tests doesn't exist yet or it is not implemented yet.

2. Write just enough code to make the test pass.

   Write the minimum amount of code necessary to pass the test. Don't add any additional features or improvements.

3. Refactor the code as needed.

   Improve design, implementation, maintainability, all the while constantly testing to make sure that all tests still pass.

Two kinds of tests

By writing tests first and then writing the code to make the tests pass, you get better code quality, improved bug detection, and a more enjoyable development experience.

Two kinds of tests

By writing tests first and then writing the code to make the tests pass, you get better code quality, improved bug detection, and a more enjoyable development experience.

Unit tests

Tests individual pieces of code, such as functions or methods, in isolation.

Two kinds of tests

By writing tests first and then writing the code to make the tests pass, you get better code quality, improved bug detection, and a more enjoyable development experience.

Unit tests

Tests individual pieces of code, such as functions or methods, in isolation.

Integration tests

Tests how multiple pieces of code work together.

TDD in practice: An example

- Consider a simple example of a function that performs a statistical calculation (e.g., linear regression).

TDD in practice: An example

- Consider a simple example of a function that performs a statistical calculation (e.g., linear regression).
- Before writing any production code, write test cases that test the expected behavior of the function

TDD in practice: An example

- Consider a simple example of a function that performs a statistical calculation (e.g., linear regression).
- Before writing any production code, write test cases that test the expected behavior of the function
    - Function gives correct output with known inputs.

TDD in practice: An example

- Consider a simple example of a function that performs a statistical calculation (e.g., linear regression).
- Before writing any production code, write test cases that test the expected behavior of the function
  - Function gives correct output with known inputs.
  - Function correctly verifies input (throwing errors for bad input is desirable behavior!)

TDD in practice: An example

- Consider a simple example of a function that performs a statistical calculation (e.g., linear regression).
- Before writing any production code, write test cases that test the expected behavior of the function
  - Function gives correct output with known inputs.
  - Function correctly verifies input (throwing errors for bad input is desirable behavior!)
- Write the function, making sure it passes all the test cases.

TDD in practice: An example

- Consider a simple example of a function that performs a statistical calculation (e.g., linear regression).
- Before writing any production code, write test cases that test the expected behavior of the function
  - Function gives correct output with known inputs.
  - Function correctly verifies input (throwing errors for bad input is desirable behavior!)
- Write the function, making sure it passes all the test cases.
- Refactor the code as needed, making sure all tests still pass.

TDD in practice: An example

- Consider a simple example of a function that performs a statistical calculation (e.g., linear regression).
- Before writing any production code, write test cases that test the expected behavior of the function
  - Function gives correct output with known inputs.
  - Function correctly verifies input (throwing errors for bad input is desirable behavior!)
- Write the function, making sure it passes all the test cases.
- Refactor the code as needed, making sure all tests still pass.
- Repeat the process for any additional features or bug fixes.

Benefits of TDD

- TDD helps to ensure code correctness and reduce the number of bugs.

Benefits of TDD

- TDD helps to ensure code correctness and reduce the number of bugs.
- It makes it easier to make changes to code and add new features, as any breaking changes will be caught by the tests.

Benefits of TDD

- TDD helps to ensure code correctness and reduce the number of bugs.
- It makes it easier to make changes to code and add new features, as any breaking changes will be caught by the tests.
- TDD improves code maintainability by providing a clear and complete set of tests for the code.

Benefits of TDD

- TDD helps to ensure code correctness and reduce the number of bugs.
- It makes it easier to make changes to code and add new features, as any breaking changes will be caught by the tests.
- TDD improves code maintainability by providing a clear and complete set of tests for the code.
- It also helps to improve the development process by making it more incremental and iterative, allowing for a faster development cycle.

Cognitive modeling is TTD

TDD in cognitive model development

- Cognitive modeling is amateur software development.

TDD in cognitive model development

- Cognitive modeling is amateur software development.
- Like any software development, it benefits from TDD.

TDD in cognitive model development

- Cognitive modeling is amateur software development.
- Like any software development, it benefits from TDD.
- Tests verify correctness of a model's predictions and behavior.

TDD in cognitive model development

- Cognitive modeling is amateur software development.
- Like any software development, it benefits from TDD.
- Tests verify correctness of a model's predictions and behavior.
- Once TDD is implemented, we can go through rapid cycles of model development:

TDD in cognitive model development

- Cognitive modeling is amateur software development.
- Like any software development, it benefits from TDD.
- Tests verify correctness of a model's predictions and behavior.
- Once TDD is implemented, we can go through rapid cycles of model development:
    - Make it easier to make changes or add new features (to the cognitive model, not just software).

TDD in cognitive model development

- Cognitive modeling is amateur software development.
- Like any software development, it benefits from TDD.
- Tests verify correctness of a model's predictions and behavior.
- Once TDD is implemented, we can go through rapid cycles of model development:
  - Make it easier to make changes or add new features (to the cognitive model, not just software).
  - Ensure the reliability and robustness of a model.

TDD in cognitive model development

- Cognitive modeling is amateur software development.
- Like any software development, it benefits from TDD.
- Tests verify correctness of a model's predictions and behavior.
- Once TDD is implemented, we can go through rapid cycles of model development:
    - Make it easier to make changes or add new features (to the cognitive model, not just software).
    - Ensure the reliability and robustness of a model.
    - Continuous testing cycles facilitate rapid model development.

TDD in cognitive model development

Examples of tests for a cognitive model could include:

- Comparison of model predictions against observed data: does the model (still) fit? Does it now fit better, or worse?

TDD in cognitive model development

Examples of tests for a cognitive model could include:

- Comparison of model predictions against observed data: does the model (still) fit? Does it now fit better, or worse?
- Comparison of model predictions against common sense.

TDD in cognitive model development

Examples of tests for a cognitive model could include:

- Comparison of model predictions against observed data: does the model (still) fit? Does it now fit better, or worse?
- Comparison of model predictions against common sense.
- Testing the model's robustness when we change...

TDD in cognitive model development

Examples of tests for a cognitive model could include:

- Comparison of model predictions against observed data: does the model (still) fit? Does it now fit better, or worse?
- Comparison of model predictions against common sense.
- Testing the model's robustness when we change...
  - independent variables

TDD in cognitive model development

Examples of tests for a cognitive model could include:

- Comparison of model predictions against observed data: does the model (still) fit? Does it now fit better, or worse?
- Comparison of model predictions against common sense.
- Testing the model's robustness when we change...
  - independent variables
  - dependent variables

TDD in cognitive model development

Examples of tests for a cognitive model could include:

- Comparison of model predictions against observed data: does the model (still) fit? Does it now fit better, or worse?
- Comparison of model predictions against common sense.
- Testing the model's robustness when we change...
  - independent variables
  - dependent variables
  - convenience assumptions we may have made

TDD in cognitive model development

Examples of tests for a cognitive model could include:

- Comparison of model predictions against observed data: does the model (still) fit? Does it now fit better, or worse?
- Comparison of model predictions against common sense.
- Testing the model's robustness when we change...
  - independent variables
  - dependent variables
  - convenience assumptions we may have made
  - structural relationships within the model

TDD in cognitive model development

Examples of tests for a cognitive model could include:

- Comparison of model predictions against observed data: does the model (still) fit? Does it now fit better, or worse?
- Comparison of model predictions against common sense.
- Testing the model's robustness when we change...
  - independent variables
  - dependent variables
  - convenience assumptions we may have made
  - structural relationships within the model
  - add entirely new features

TDD in cognitive model development

Examples of tests for a cognitive model could include:

- Comparison of model predictions against observed data: does the model (still) fit? Does it now fit better, or worse?
- Comparison of model predictions against common sense.
- Testing the model's robustness when we change...
  - independent variables
  - dependent variables
  - convenience assumptions we may have made
  - structural relationships within the model
  - add entirely new features
  - ...

TDD in cognitive model development

Examples of tests for a cognitive model could include:

- Comparison of model predictions against observed data: does the model (still) fit? Does it now fit better, or worse?
- Comparison of model predictions against common sense.
- Testing the model's robustness when we change...
  - independent variables
  - dependent variables
  - convenience assumptions we may have made
  - structural relationships within the model
  - add entirely new features
  - ...
- Often the test can be evaluated informally (e.g., looking at a figure of model vs. data), but ideally it is automated.

TDD workflow

TDD workflow start

1. **Write a test:** Write a test that defines a desired behavior or feature of the code you are developing.

TDD workflow start

1. **Write a test:** Write a test that defines a desired behavior or feature of the code you are developing.
2. **Watch it fail:** Run the test and see that it fails, as the desired behavior or feature has not been implemented yet.

TDD workflow start

1. **Write a test:** Write a test that defines a desired behavior or feature of the code you are developing.
2. **Watch it fail:** Run the test and see that it fails, as the desired behavior or feature has not been implemented yet.
3. **Write the code:** Write the minimum amount of code needed to make the test pass.

TDD workflow start

1. **Write a test:** Write a test that defines a desired behavior or feature of the code you are developing.
2. **Watch it fail:** Run the test and see that it fails, as the desired behavior or feature has not been implemented yet.
3. **Write the code:** Write the minimum amount of code needed to make the test pass.
4. **Watch it pass:** Run the test and see that it passes, as the desired behavior or feature has now been implemented.

TDD workflow start

1. **Write a test:** Write a test that defines a desired behavior or feature of the code you are developing.
2. **Watch it fail:** Run the test and see that it fails, as the desired behavior or feature has not been implemented yet.
3. **Write the code:** Write the minimum amount of code needed to make the test pass.
4. **Watch it pass:** Run the test and see that it passes, as the desired behavior or feature has now been implemented.
5. **Refactor:** Refactor the code as needed, making sure all tests continue to pass.

TDD workflow continue

6. **Repeat the process:** Repeat the process of adding a test, running all tests, writing the code, running all tests, and refactoring as needed for each desired behavior or feature.

TDD workflow continue

6. **Repeat the process:** Repeat the process of adding a test, running all tests, writing the code, running all tests, and refactoring as needed for each desired behavior or feature.
7. **Continuously verify:** Continuously verify the correctness of the code through the tests.

TDD workflow continue

6. **Repeat the process:** Repeat the process of adding a test, running all tests, writing the code, running all tests, and refactoring as needed for each desired behavior or feature.
7. **Continuously verify:** Continuously verify the correctness of the code through the tests.
8. **Maintaining a test suite:** Maintaining a comprehensive test suite that covers all desired behaviors and features of the code.

Unit tests

Unit tests in MATLAB

```
function tests = test_example
  tests = functiontests(localfunctions);
end
  function testSin(testCase)  % Test that sin(pi/2) is 1
    obtainedAnswer = sin(pi/2);
    expectedAnswer = 1;
    verifyEqual(testCase,obtainedAnswer,expectedAnswer)
  end
  function testRoots(testCase)  % Test that roots are 2
    obtainedAnswer = roots([1 -4 4]);
    expectedAnswer = [2 2];
    verifyEqual(testCase,obtainedAnswer,expectedAnswer)
  end
```

Unit tests in MATLAB

```
function tests = test_example
  tests = functiontests(localfunctions);
end
  function testSin(testCase)  % Test that sin(pi/2) is 1
    obtainedAnswer = sin(pi/2);
    expectedAnswer = 1;
    verifyEqual(testCase,obtainedAnswer,expectedAnswer)
  end
  function testRoots(testCase)  % Test that roots are 2
    obtainedAnswer = roots([1 -4 4]);
    expectedAnswer = [2 2];
    verifyEqual(testCase,obtainedAnswer,expectedAnswer)
  end
```

```
suite = testSuite({'test_example'});
result = run(suite);
disp(table(result))
if all([result.Passed])
    disp('All tests passed.')
else disp('There were test failures.')
end
```

Unit tests in MATLAB

More on unit tests in MATLAB via The MathWorks:

https://www.mathworks.com/help/matlab/matlab_prog/write-function-based-unit-tests.html

https://www.mathworks.com/help/matlab/matlab_prog/write-simple-test-case-with-functions.html

An applied example

Bayes factor for binomial data

- Let's consider a binomial experiment with n trials and k successes.

Bayes factor for binomial data

- Let's consider a binomial experiment with n trials and k successes.
- Let's denote the success probability by θ.

Bayes factor for binomial data

- Let's consider a binomial experiment with n trials and k successes.
- Let's denote the success probability by θ.
- We have two competing models that differ in their prior distributions for θ:

Bayes factor for binomial data

- Let's consider a binomial experiment with n trials and k successes.
- Let's denote the success probability by θ.
- We have two competing models that differ in their prior distributions for θ:
  - The "slab" prior: $\theta \sim U(0, 1)$

Bayes factor for binomial data

- Let's consider a binomial experiment with n trials and k successes.
- Let's denote the success probability by θ.
- We have two competing models that differ in their prior distributions for θ:
  - The "slab" prior: $\theta \sim U(0, 1)$
  - The "spike" prior: $\theta \sim U(0.45, 0.55)$

Bayes factor for binomial data

- The posterior distributions given the data can be found as:

$$
\begin{aligned}
p(\theta|k) &\propto p(k|\theta)p(\theta) \\
p(\theta|k, \text{slab}) &\propto \binom{n}{k}\theta^k(1-\theta)^{n-k} \times U(\theta|0,1) \\
p(\theta|k, \text{spike}) &\propto \binom{n}{k}\theta^k(1-\theta)^{n-k} \times U(\theta|0.45, 0.55)
\end{aligned}
$$

Bayes factor for binomial data

- The posterior distributions given the data can be found as:

$$
\begin{aligned}
p(\theta|k) &\propto p(k|\theta)p(\theta) \\
p(\theta|k, \text{slab}) &\propto \binom{n}{k}\theta^k(1-\theta)^{n-k} \times U(\theta|0, 1) \\
p(\theta|k, \text{spike}) &\propto \binom{n}{k}\theta^k(1-\theta)^{n-k} \times U(\theta|0.45, 0.55)
\end{aligned}
$$

- The Bayes factor for comparing the two models can be calculated as:

$$
\begin{aligned}
B &= \frac{p(k|\text{spike})}{p(k|\text{slab})} \\
&= \frac{\int_{0.45}^{0.55} \binom{n}{k}\theta^k(1-\theta)^{n-k} d\theta}{\int_0^1 \binom{n}{k}\theta^k(1-\theta)^{n-k} d\theta}
\end{aligned}
$$