universität wien

FWF

TTEST for Matlab/Octave

Unit testing scientific software

2021-10-20

Clara Hollomey / Thomas Mejstrik

```
EXPECT_EQ( TTEST, 'Great' )
```

`EXPECT_EQ( TTEST, 'Great' )`

TTEST - Unit test framework for Matlab/Octave

- Not a math talk (but rather a social science talk)

Please ask questions.

`EXPECT_EQ( TTEST, 'Great' )`

TTEST - Unit test framework for Matlab/Octave

- Not a math talk (but rather a software design talk)
- Test driven development *(TDD)*

Please ask questions.

EXPECT_EQ(TTEST, 'Great')

TTEST - Unit test framework for Matlab/Octave

- Not a math talk (but rather a software design talk)
- Test driven development *(TDD)*
- *TTEST* / Considerations about unit test frameworks

Please ask questions.

`EXPECT_EQ( TTEST, 'Great' )`

TTEST - Unit test framework for Matlab/Octave

- Not a math talk (but rather a software design talk)
- Test driven development *(TDD)*
- *TTEST* / Considerations about unit test frameworks
- Matlab Hacks

Please ask questions.

Test driven development *(TDD)*

TDD - Unit test

What are unit tests?

Unit tests are typically automated tests to ensure that a section of an application (known as the "unit") meets its design and behaves as intended.[1]

[1] Paul Hamill, *Unit Test Frameworks: Tools for High-Quality Software Development*, 2004.

TDD - Unit test

What are unit tests?

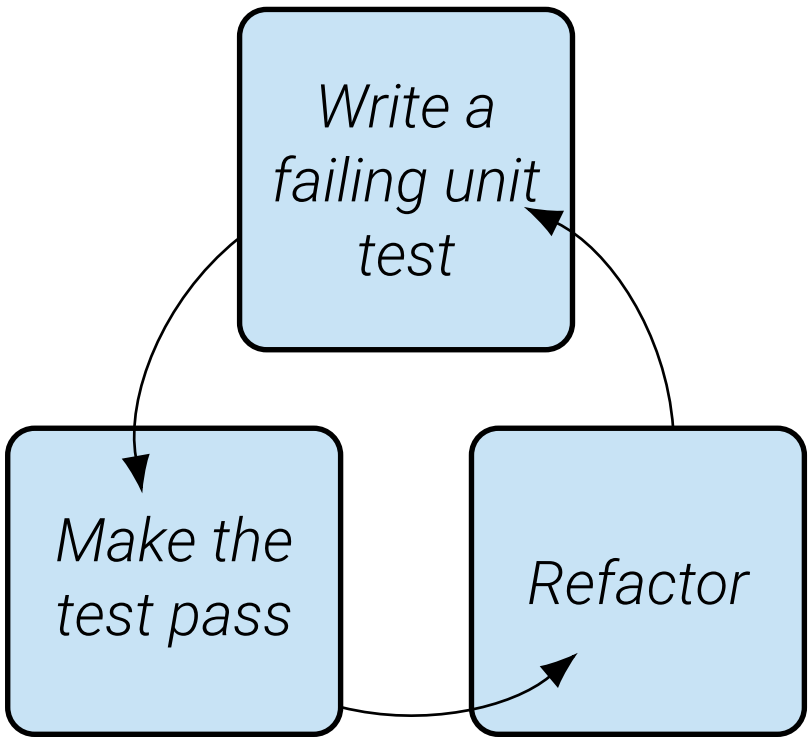
Unit tests are typically automated tests to ensure that a section of an application (known as the "unit") meets its design and behaves as intended.[1]

Example:

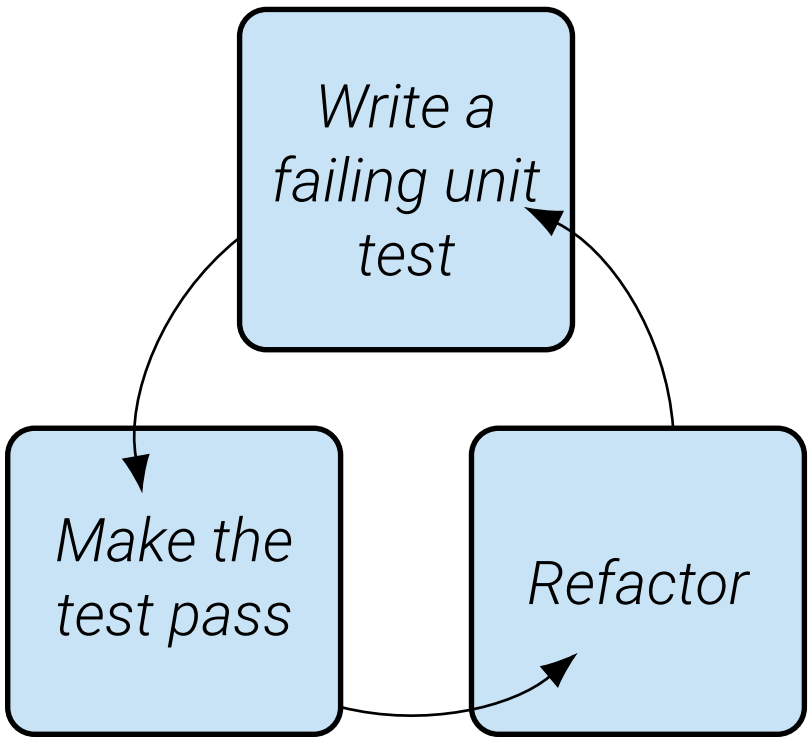
```
EXPECT_EQ( eig([2 2;1 1]), [3;0] );
    % not yet automated
```

[1] Paul Hamill, *Unit Test Frameworks: Tools for High-Quality Software Development*, 2004.

TDD - The cycle (Half of the talk on one slide)



TDD - The cycle (Half of the talk on one slide)



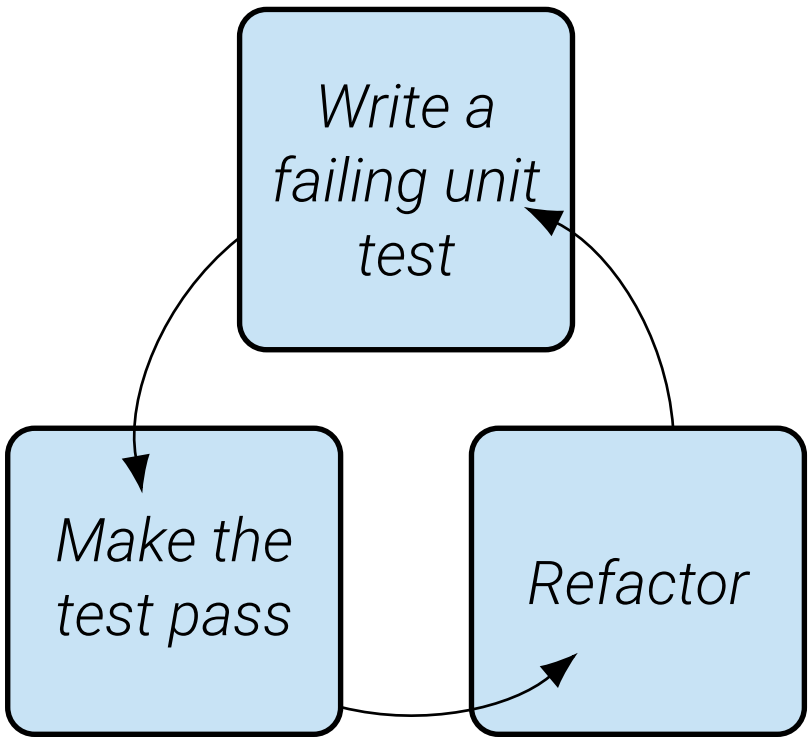
Benefits of TDD:

- [ ] Increases code coverage
- [ ] Saves time (on the long term)
- [ ] Forces good interfaces
- [ ] Provides documentation
- [ ] Helps to find bugs
- [ ] Prevents from regression
- [ ] Simplifies debugging

Design of the framework:

- [ ] Easy to use
- [ ] Helpful
- [ ] Fast

TDD - The cycle (Half of the talk on one slide)



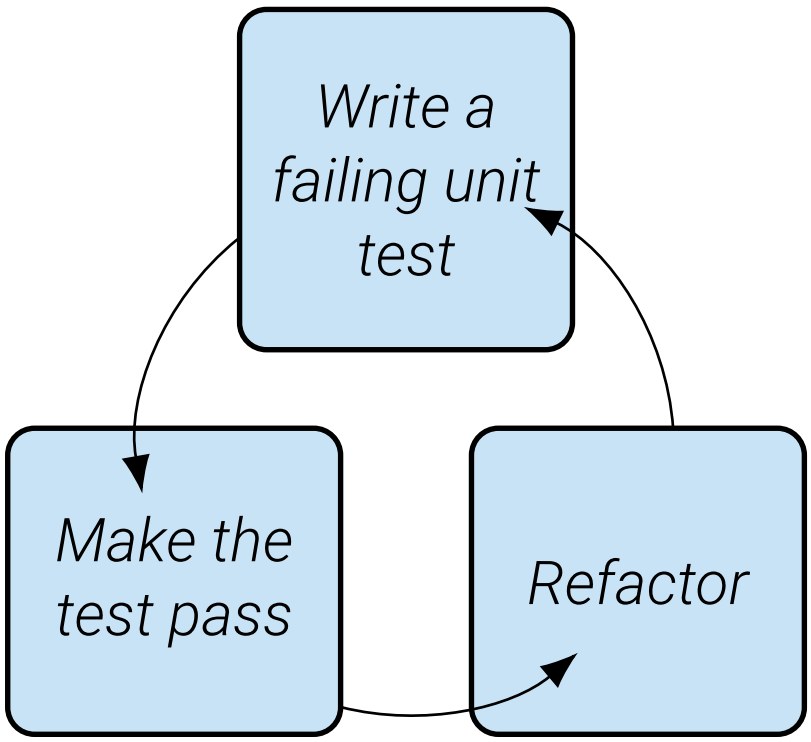
Benefits of TDD:

- Increases code coverage
- Saves time (on the long term)
- Forces good interfaces
- Provides documentation
- Helps to find bugs
- Prevents from regression
- Simplifies debugging

Design of the framework:

- Easy to use
- Helpful
- Fast

TDD - The cycle (Half of the talk on one slide)



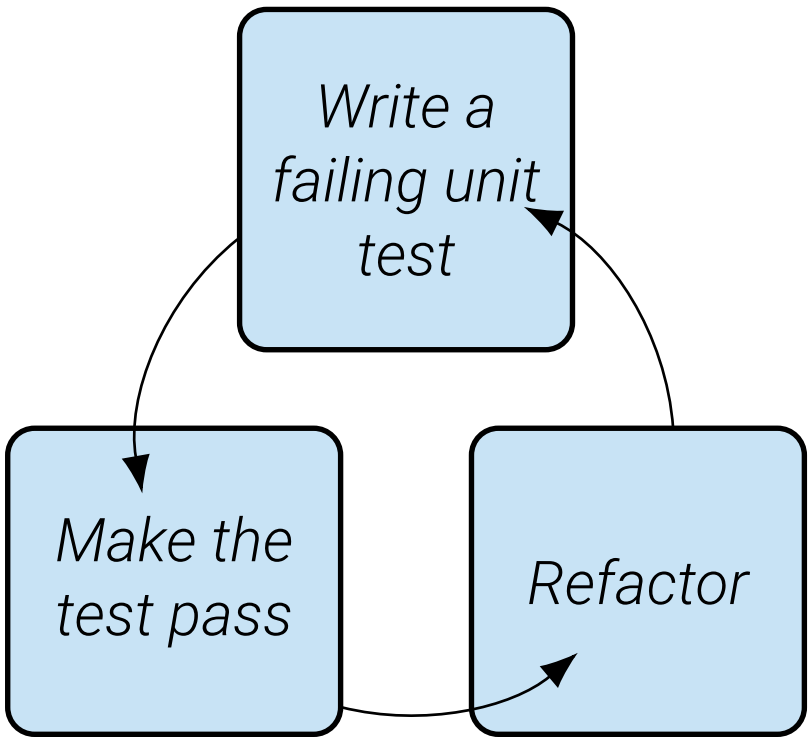
Benefits of TDD:

- [ ] Increases code coverage
- [ ] Saves time (on the long term)
- [ ] Forces good interfaces
- [ ] Provides documentation
- [ ] Helps to find bugs
- [ ] Prevents from regression
- [ ] Simplifies debugging

Design of the framework:

- [ ] Easy to use
- [ ] Helpful
- [ ] Fast

TDD - The cycle (Half of the talk on one slide)



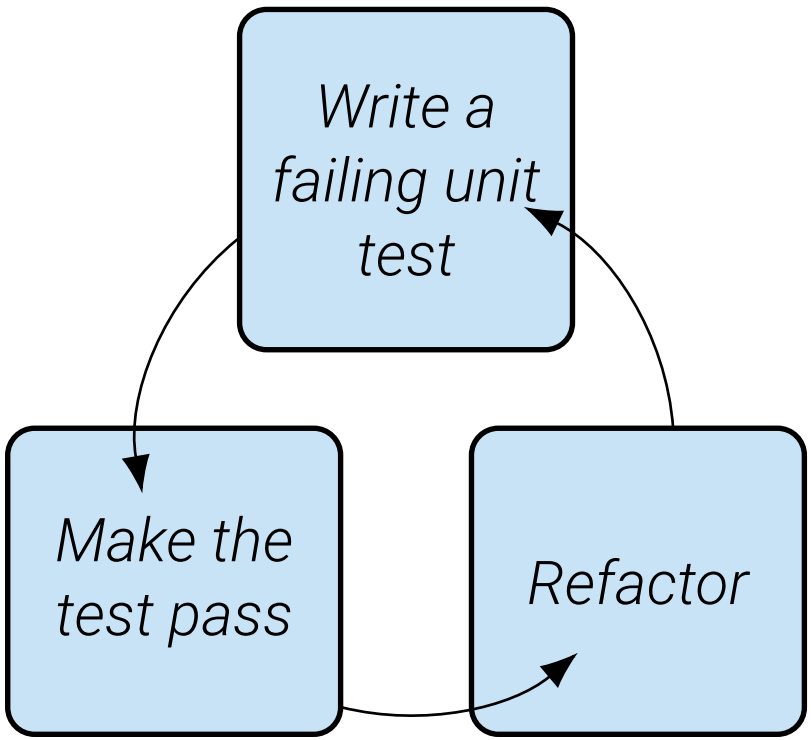
Benefits of TDD:

- [ ] Increases code coverage
- [ ] Saves time (on the long term)
- [ ] Forces good interfaces
- [ ] Provides documentation
- [ ] Helps to find bugs
- [ ] Prevents from regression
- [ ] Simplifies debugging

Design of the framework:

- [ ] Easy to use
- [ ] Helpful
- [ ] Fast

TDD - The cycle (Half of the talk on one slide)



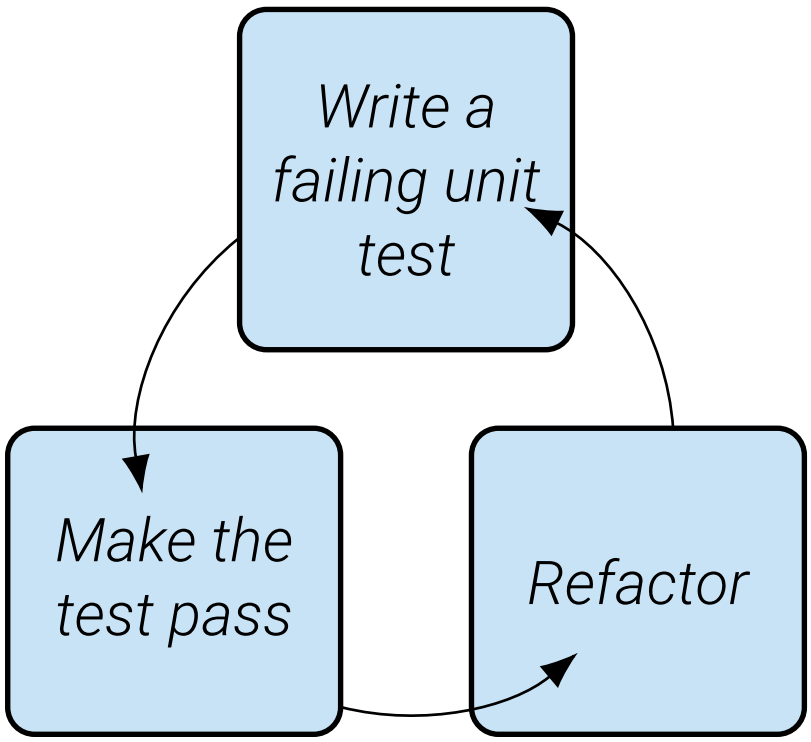
Benefits of TDD:

- Increases code coverage
- Saves time (on the long term)
- Forces good interfaces
- Provides documentation
- Helps to find bugs
- Prevents from regression
- Simplifies debugging

Design of the framework:

- Easy to use
- Helpful
- Fast

TDD - The cycle (Half of the talk on one slide)



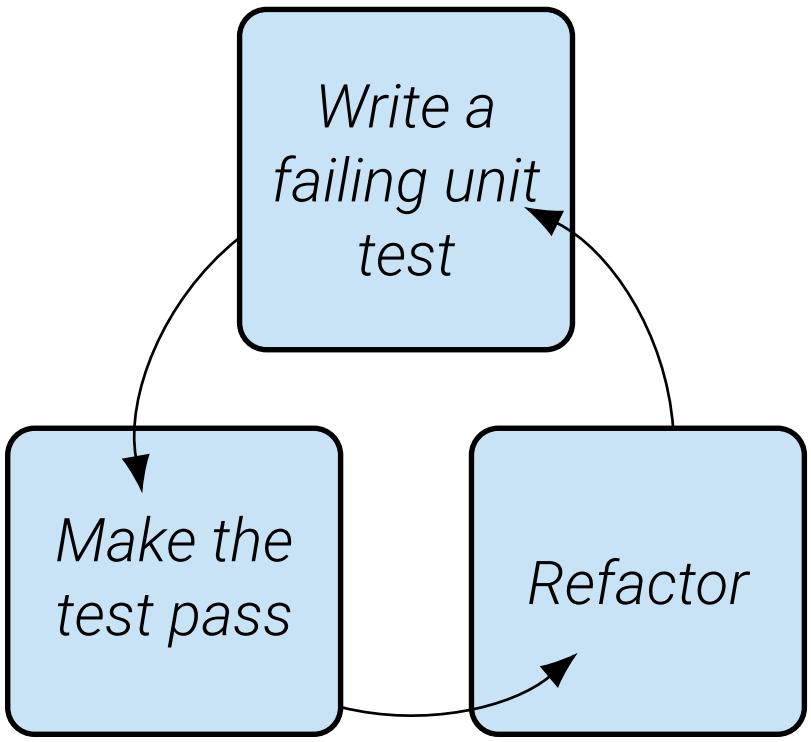
Benefits of TDD:

- Increases code coverage
- Saves time (on the long term)
- Forces good interfaces
- Provides documentation
- Helps to find bugs
- Prevents from regression
- Simplifies debugging

Design of the framework:

- Easy to use
- Helpful
- Fast

TDD - The cycle (Half of the talk on one slide)



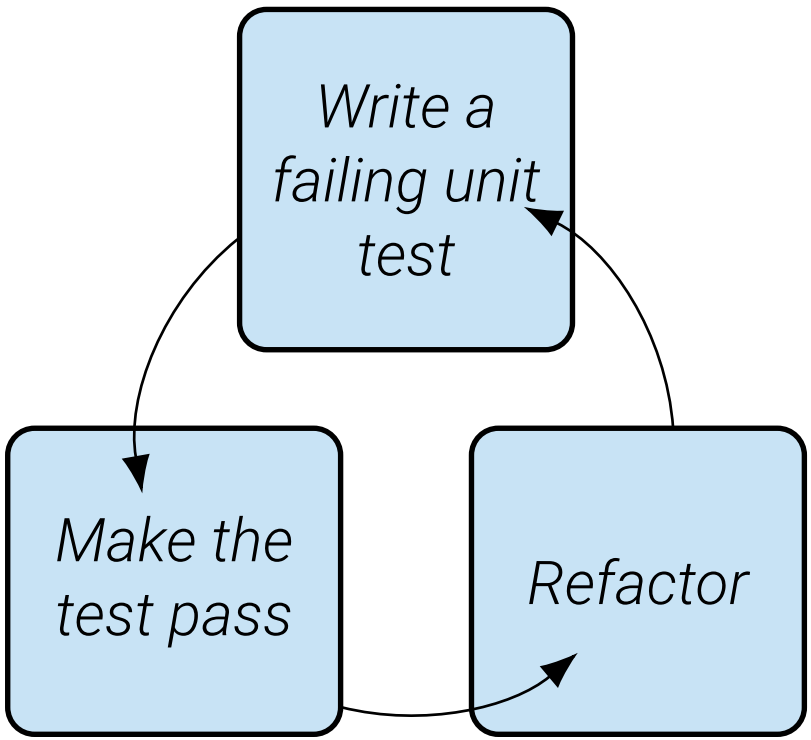
Benefits of TDD:

- Increases code coverage
- Saves time (on the long term)
- Forces good interfaces
- Provides documentation
- Helps to find bugs
- Prevents from regression
- Simplifies debugging

Design of the framework:

- Easy to use
- Helpful
- Fast

TDD - The cycle (Half of the talk on one slide)



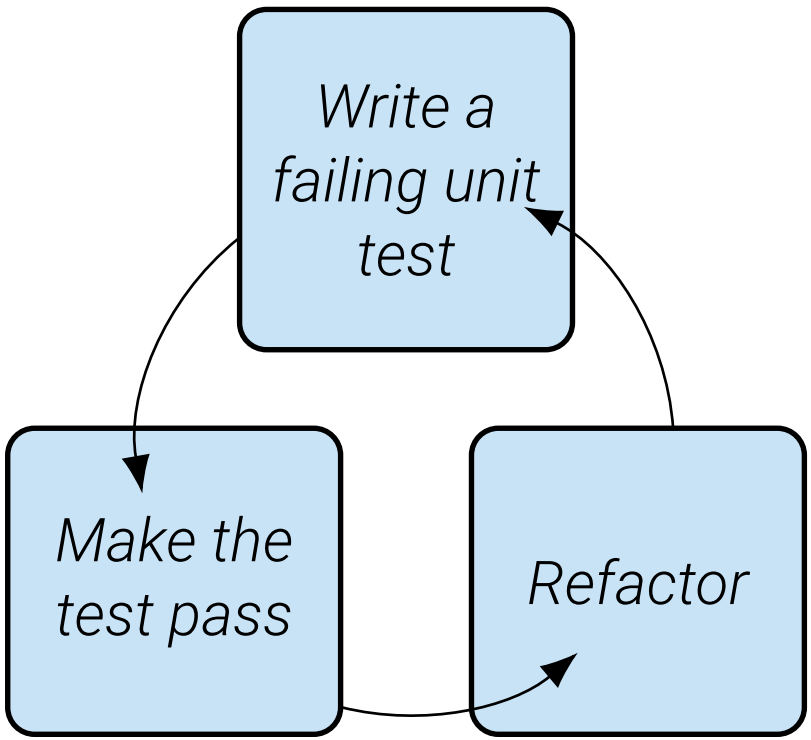
Benefits of TDD:

- Increases code coverage
- Saves time (on the long term)
- Forces good interfaces
- Provides documentation
- Helps to find bugs
- Prevents from regression
- Simplifies debugging

Design of the framework:

- Easy to use
- Helpful
- Fast

TDD - The cycle (Half of the talk on one slide)



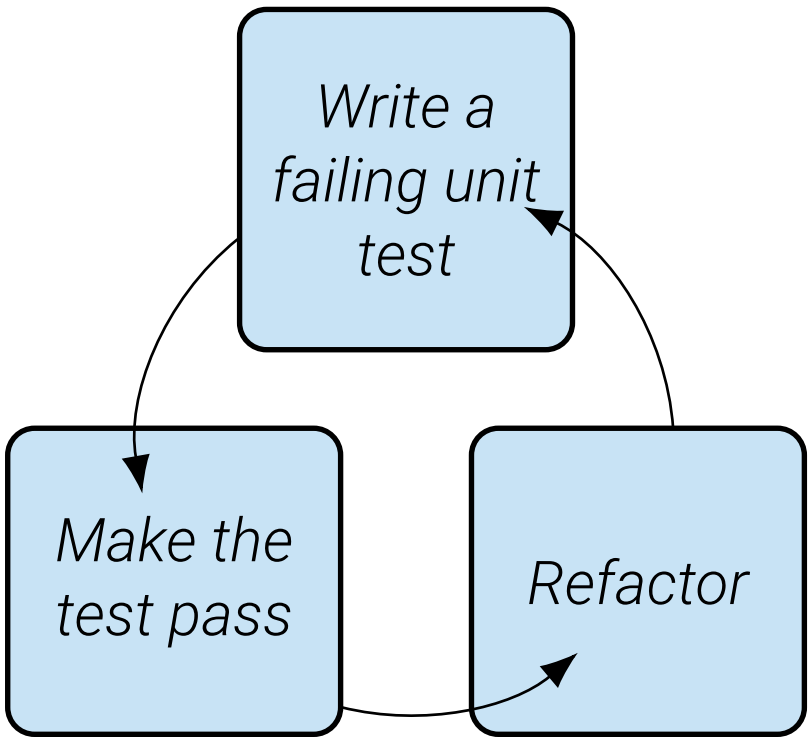
Benefits of TDD:

- Increases code coverage
- Saves time (on the long term)
- Forces good interfaces
- Provides documentation
- Helps to find bugs
- Prevents from regression
- Simplifies debugging

Design of the framework:

- Easy to use
- Helpful
- Fast

TDD - The cycle (Half of the talk on one slide)



Benefits of TDD:

- Increases code coverage
- Saves time (on the long term)
- Forces good interfaces
- Provides documentation
- Helps to find bugs
- Prevents from regression
- Simplifies debugging

Design of the framework:

- Easy to use
- Helpful
- Fast

Scientific Context

TDD - Scientific context

- People who use Matlab are, to a large extent, not developers but mathematicians or engineers.[2]

[2] Conversely, most developers do not use Matlab.

TDD - Scientific context

- People who use Matlab are, to a large extent, not developers but mathematicians or engineers.[2]
- High fluctuation of responsible personnel

[2] Conversely, most developers do not use Matlab.

TDD - Scientific context

- People who use Matlab are, to a large extent, not developers but mathematicians or engineers.[2]
- High fluctuation of responsible personnel
- People work often on their own. Usually there is no code review.

[2] Conversely, most developers do not use Matlab.

TDD - Scientific context

- People who use Matlab are, to a large extent, not developers but mathematicians or engineers.[2]
- High fluctuation of responsible personnel
- People work often on their own. Usually there is no code review.
- Exact replication of results is important

[2] Conversely, most developers do not use Matlab.

TDD - Basic Unit test framework design

SUnit $\mapsto$ JUnit $\mapsto$ SUnit $\mapsto$ {xUnit}[3,4]

- *SUT* (System under test)
- *Test runner*
- *Test fixture*
- *Test suite / Test cases*
- *Assertions*

Junit for Java *Catch2*, *GoogleTest* for C++ *PyTest*, *unittest*, *Hypothesis* for Python *Julia's built in unit test framework* for Julia *D's built in unit test framework* for D *testthat* for R *pFUnit* for Fortran *MOxUnit*, *xunit4*, *Matlab's built in unit test framework* for Matlab

Some more Matlab frameworks *Check*, *CppUTest*, *CUnit*, *CUT*, *MinUnit*, *Theft* for C *Bandit*, *nanobench*, *Google Benchmark*, *Boost Test Library*, *CppUnit*, *CUTE*, *doctest* and *Fructose* for C++ *pytruth* for Python *NUnit* for .NET *QuickCheck* for Haskell

---

[3] Kent Beck, *Simple Smalltalk Testing: With Patterns*, 1989.

[4] Even Kent Beck does not always do *tests first*.

TDD - Basic Unit test framework design

SUnit $\mapsto$ JUnit $\mapsto$ SUnit $\mapsto$ {xUnit}[3,4]

- *SUT* (System under test)
- *Test runner*
- *Test fixture*
- *Test suite / Test cases*
- *Assertions*

Junit for Java *Catch2*, *GoogleTest* for C++ *PyTest*, *unittest*, *Hypothesis* for Python *Julia's built in unit test framework* for Julia *D's built in unit test framework* for D *testthat* for R *pFUnit* for Fortran *MOxUnit*, *xunit4*, *Matlab's built in unit test framework* for Matlab

Some more Matlab frameworks *Check*, *CppUTest*, *CUnit*, *CUT*, *MinUnit*, *Theft* for C *Bandit*, *nanobench*, *Google Benchmark*, *Boost Test Library*, *CppUnit*, *CUTE*, *doctest* and *Fructose* for C++ *pytruth* for Python *NUnit* for .NET *QuickCheck* for Haskell

---

[3] Kent Beck, *Simple Smalltalk Testing: With Patterns*, 1989.

[4] Even Kent Beck does not always do *tests first*.

TDD - Basic Unit test framework design

SUnit $\mapsto$ JUnit $\mapsto$ SUnit $\mapsto$ {xUnit}[3,4]

- *SUT* (System under test)
- *Test runner*
- *Test fixture*
- *Test suite / Test cases*
- *Assertions*

Junit for Java *Catch2*, *GoogleTest* for C++ *PyTest*, *unittest*, *Hypothesis* for Python *Julia's built in unit test framework* for Julia *D's built in unit test framework* for D *testthat* for R *pFUnit* for Fortran *MOxUnit*, *xunit4*, *Matlab's built in unit test framework* for Matlab

Some more Matlab frameworks *Check*, *CppUTest*, *CUnit*, *CUT*, *MinUnit*, *Theft* for C *Bandit*, *nanobench*, *Google Benchmark*, *Boost Test Library*, *CppUnit*, *CUTE*, *doctest* and *Fructose* for C++ *pytruth* for Python *NUnit* for .NET *QuickCheck* for Haskell

---

[3] Kent Beck, *Simple Smalltalk Testing: With Patterns*, 1989.

[4] Even Kent Beck does not always do *tests first*.

TDD - Basic Unit test framework design

SUnit $\mapsto$ JUnit $\mapsto$ SUnit $\mapsto$ {xUnit}[3,4]

- *SUT* (System under test)
- *Test runner*
- *Test fixture*
- *Test suite / Test cases*
- *Assertions*

Junit for Java *Catch2*, *GoogleTest* for C++ *PyTest*, *unittest*, *Hypothesis* for Python *Julia's built in unit test framework* for Julia *D's built in unit test framework* for D *testthat* for R *pFUnit* for Fortran *MOxUnit*, *xunit4*, *Matlab's built in unit test framework* for Matlab

Some more Matlab frameworks *Check*, *CppUTest*, *CUnit*, *CUT*, *MinUnit*, *Theft* for C *Bandit*, *nanobench*, *Google Benchmark*, *Boost Test Library*, *CppUnit*, *CUTE*, *doctest* and *Fructose* for C++ *pytruth* for Python *NUnit* for .NET *QuickCheck* for Haskell

---

[3] Kent Beck, *Simple Smalltalk Testing: With Patterns*, 1989.

[4] Even Kent Beck does not always do *tests first*.

TDD - Basic Unit test framework design

SUnit $\mapsto$ JUnit $\mapsto$ SUnit $\mapsto$ {xUnit}[3,4]

- *SUT* (System under test)
- *Test runner*
- *Test fixture*
- *Test suite / Test cases*
- *Assertions*

Junit for Java *Catch2*, *GoogleTest* for C++ *PyTest*, *unittest*, *Hypothesis* for Python *Julia's built in unit test framework* for Julia *D's built in unit test framework* for D *testthat* for R *pFUnit* for Fortran *MOxUnit*, *xunit4*, *Matlab's built in unit test framework* for Matlab

Some more Matlab frameworks *Check*, *CppUTest*, *CUnit*, *CUT*, *MinUnit*, *Theft* for C *Bandit*, *nanobench*, *Google Benchmark*, *Boost Test Library*, *CppUnit*, *CUTE*, *doctest* and *Fructose* for C++ *pytruth* for Python *NUnit* for .NET *QuickCheck* for Haskell

---

[3] Kent Beck, *Simple Smalltalk Testing: With Patterns*, 1989.

[4] Even Kent Beck does not always do *tests first*.

TDD - Matlab unit test frameworks

- Exhaustive List (most likely)

Name	Last Update	Name	Last Update
Kit Ng *Unit testing tools*	2005	Legland *munit*	2016
Phelan *MUnit*	2006	Schiessl *unitTAPsci*	2016
Lombardi *MUnit*	2006	mwgeurts *unit test harness*	2016
Kritzinger *Test Framework*	2006	Sexton *xunit4*	2016
Christopher *mlunit 2008a*	2009	Bergholz *mUnittest*	2017
Smith *Doctest*	2010	*MOxUnit*	2021
Eddins *xUnit*	2010	Zimmermann *MP-Test*	2021
Brett *matlabtesting*	2011	*Matlab's built in framework*	
Nievinski *testit*	2011	Script based tests	2021
arka *mUnit*	2013	Function based tests	2021
Zyndric *lute*	2013	Class based tests	2021
Hetu *mlUnit*	2015	*TTEST*	2021

TTEST
Considerations about unit test frameworks

Usability

Considerations - Usability - Coding style

- Language proficiency: *To write a test should not require more knowledge than what is needed to write a program.*

  ~~*regular expressions*~~, ~~*tables*~~, ~~*structs*~~, ~~*classes*~~, ~~*functional programming*~~, ~~*handles*~~, ~~*error handling*~~, ~~*cell arrays*~~, ~~*function handles*~~

- ~~Function handles~~:

```
EXPECT_NTHROW( 'inv(1)' );     % as evalable string
EXPECT_NTHROW( @() inv(1) );   % as nullary lambda
```

- Aliases

Considerations - Usability - Coding style

- Language proficiency: *To write a test should not require more knowledge than what is needed to write a program.*

  ~~*regular expressions*~~, ~~*tables*~~, ~~*structs*~~, ~~*classes*~~, ~~*functional programming*~~, ~~*handles*~~, ~~*error handling*~~, ~~*cell arrays*~~, ~~*function handles*~~

- ~~Function handles~~:

```
EXPECT_NTHROW( 'inv(1)' );     % as evalable string
EXPECT_NTHROW( @() inv(1) );   % as nullary lambda
```

- Aliases

Considerations - Usability - Coding style

- Language proficiency: *To write a test should not require more knowledge than what is needed to write a program.*

  ~~*regular expressions*~~, ~~*tables*~~, ~~*structs*~~, ~~*classes*~~, ~~*functional programming*~~, ~~*handles*~~, ~~*error handling*~~, ~~*cell arrays*~~, ~~*function handles*~~

- ~~Function handles~~:

```
EXPECT_NTHROW( 'inv(1)' );     % as evalable string
EXPECT_NTHROW( @() inv(1) );   % as nullary lambda
```

- Aliases

Considerations - Usability - Performance

	(E)	*(1024)*	*(16)*
TTEST Script	0.69 *s*	9.1 *s*	16 *s*
TTEST Function	0.76 *s*	5.5 *s*	4.9 *s*
MOxUnit	0.002 *s*	9.4 *s*	23 *s*
xunit4	0.001 *s*	2.5 *s*	4.8 *s*
Matlab's framework Script	0.76 *s*	40 *s*	140 *s*
Matlab's framework Function	0.13 *s*	31 *s*	105 *s*
Matlab's framework Class	0.14 *s*	210 *s*	94 *s*

Considerations - Test suite - Excess Code

- *GoogleTest* for C++

```
#include <gtest.h>
TEST( Test2, Test22 ) {
  EXPECT_EQ( 2, 2 );
}
```

- *Matlab's framework* Script with test runner `runtests`

```
%% test_2_2
  assert( 2 == 2 );
```

- *TTEST* script based with test runner `runtests`

```
%% test_2_2
  EXPECT_EQ( 2, 2 );
```

- *TTEST* script based with test runner `runttests`

```
TTEST init
TESTCASE( 'test_2_2' )
  EXPECT_EQ( 2, 2 );
```

Considerations - Test suite - Excess Code

- *MOxUnit* with test runner `moxunit_run_tests`

```
function test_suite = test1Test
  test_functions = localfunctions();
  initTestSuite;
function test_2_2
  assertEqual( 2, 2 );
```

- *xunit4* with test runner `runxunit` (Similar to *MOxUnit*)

- *Matlab' framework* function based with test runner `runtests`

```
function tests = test2
  tests = functiontests(localfunctions);
function test_2_2( testCase )
  verifyEqual( testCase, 2, 2 );
```

- *TTEST* function based with test runner `runttests`

```
function test_2_2
  EXPECT_EQ( 2, 2 );
```

Considerations - Test suite - Excess Code

- *PyTest* for Python

```
import unittest
class test_2_2( unittest.TestCase ):
  def test_2_2( self )
    self.assertEqual( 2, 2 )
```

- *Matlab's framework* Class with test runner `runtests`

```
classdef test2 < matlab.unittest.TestCase
  methods( Test )
    function test_2_2( testCase );
      testCase.verifyEqual( 2, 2 );
    end
  end
end
```

Considerations - Test suite - Excess Code

- *Doctests* for Python

```
def add( a, b ):
  """
  >>> add(2, 3)
  5
  """
  return a + b
```

- *TTEST* doctest with test runner `runttests`

```
function x = add( a, b )
%TT EXPECT_EQ( add(2,3), 5 );
  x = a + b;
```

Test fixtures

Considerations - Test fixture

- Test independence (Sections)[5]

```
TTEST init;  % create TTEST environment
a = 1;
TESTCASE;
  % a==1
  b = 3;
  SECTION;
    % a==1, b==3
    b = 4; c = 4;
  SECTION;
    % a==1, b==3, c is undefined
TESTCASE;
  % a==1, b and c are undefined
```

[5] Martin Hořeňovský, et al., *Catch2*.

Considerations - Test fixture

What to restore	*Matlab's framework*	*MOxUnit, xunit4*	*TTEST*
workspace (variables)	Yes	Yes	Yes
base workspace	×	×	Opt.
global variables	×	×	Yes
persistent variables	×	×	×
Matlab path	×	×	Yes
working directory	×	×	Yes
namespace `imports`	Yes	×	×
global RNG	×	×	Opt.
user-defined RNG	×	×	×
debugger breakpoints	×	×	Yes
warning state	×	×	Yes
figures	×	×	×
TTEST code injections	N.A.	N.A.	Yes
packages (Octave only)	N.A.	×	×

Assertions

Considerations - Assertions

- Large set of assertions: `FAIL, ..., TRUE, ..., PRED, ..., EQ, ..., LE, ..., ALMOST_EQ, ..., NEAR, ..., RANGE, PLUSZERO, ..., STRCONTAINS, FILE_EQ, SUBSET,..., MINTIME, ..., THROW, ..., TOOLBOX, ...`
- Test macros and matchers:

```
EXPECT_THAT( 'TTEST', HasSubstr('TEST') );
EXPECT_STRCONTAINS( 'TTEST', 'TEST' );
```

- Severity levels:

```
TODO_EQ( 2, 3 );
EXPECT_EQ( DISABLED('onOctave'), 2, 3 );
ASSERT_EQ( 2, ENABLED(ismatlab), 3 );
```

- Variadic assertions:

```
EXPECT_LE( 1, 2, 3, 4 );
EXPECT_NEAR( 10, 11, 9, 10.5, 2 );
  % last argument is maximum difference
EXPECT_STRCONTAINS( @() fprintf('abc'), 'bc' );
```

Considerations - Assertions

- Large set of assertions: `FAIL, ..., TRUE, ..., PRED, ..., EQ, ..., LE, ..., ALMOST_EQ, ..., NEAR, ..., RANGE, PLUSZERO, ..., STRCONTAINS, FILE_EQ, SUBSET,..., MINTIME, ..., THROW, ..., TOOLBOX, ...`
- Test macros and matchers:

```
EXPECT_THAT( 'TTEST', HasSubstr('TEST') );
EXPECT_STRCONTAINS( 'TTEST', 'TEST' );
```

- Severity levels:

```
TODO_EQ( 2, 3 );
EXPECT_EQ( DISABLED('onOctave'), 2, 3 );
ASSERT_EQ( 2, ENABLED(ismatlab), 3 );
```

- Variadic assertions:

```
EXPECT_LE( 1, 2, 3, 4 );
EXPECT_NEAR( 10, 11, 9, 10.5, 2 );
  % last argument is maximum difference
EXPECT_STRCONTAINS( @() fprintf('abc'), 'bc' );
```

Considerations - Assertions

- Large set of assertions: `FAIL`, ..., `TRUE`, ..., `PRED`, ..., `EQ`, ..., `LE`, ..., `ALMOST_EQ`, ..., `NEAR`, ..., `RANGE`, `PLUSZERO`, ..., `STRCONTAINS`, `FILE_EQ`, `SUBSET`,..., `MINTIME`, ..., `THROW`, ..., `TOOLBOX`, ...
- Test macros and matchers:

```
EXPECT_THAT( 'TTEST', HasSubstr('TEST') );
EXPECT_STRCONTAINS( 'TTEST', 'TEST' );
```

- Severity levels:

```
TODO_EQ( 2, 3 );
EXPECT_EQ( DISABLED('onOctave'), 2, 3 );
ASSERT_EQ( 2, ENABLED(ismatlab), 3 );
```

- Variadic assertions:

```
EXPECT_LE( 1, 2, 3, 4 );
EXPECT_NEAR( 10, 11, 9, 10.5, 2 );
  % last argument is maximum difference
EXPECT_STRCONTAINS( @() fprintf('abc'), 'bc' );
```

Considerations - Assertions

- Large set of assertions: `FAIL`, ..., `TRUE`, ..., `PRED`, ..., `EQ`, ..., `LE`, ..., `ALMOST_EQ`, ..., `NEAR`, ..., `RANGE`, `PLUSZERO`, ..., `STRCONTAINS`, `FILE_EQ`, `SUBSET`,..., `MINTIME`, ..., `THROW`, ..., `TOOLBOX`, ...
- Test macros and matchers:

```
EXPECT_THAT( 'TTEST', HasSubstr('TEST') );
EXPECT_STRCONTAINS( 'TTEST', 'TEST' );
```

- Severity levels:

```
TODO_EQ( 2, 3 );
EXPECT_EQ( DISABLED('onOctave'), 2, 3 );
ASSERT_EQ( 2, ENABLED(ismatlab), 3 );
```

- Variadic assertions:

```
EXPECT_LE( 1, 2, 3, 4 );
EXPECT_NEAR( 10, 11, 9, 10.5, 2 );
  % last argument is maximum difference
EXPECT_STRCONTAINS( @() fprintf('abc'), 'bc' );
```

Considerations - Assertions - PBT / Gold standards

- Property based testing *(PBT)*[6]

```
GIVEN( @(mat) isequaln(mat.', tp(mat)) );
GIVEN( matrix, @(x) isequaln(x.', tp(x)) );
```

- Gold standard tests

```
EXPECT_EQ( CACHE('name'), 2 );
```

  - Highly complex truth
  - Unknown truth

[6] David R. MacIver, et al., *Hypothesis*.

Considerations - Assertions - PBT / Gold standards

- Property based testing *(PBT)*[6]

```
GIVEN( @(mat) isequaln(mat.', tp(mat)) );
GIVEN( matrix, @(x) isequaln(x.', tp(x)) );
```

- Gold standard tests

```
EXPECT_EQ( CACHE('name'), 2 );
```

  - Highly complex truth
  - Unknown truth

[6] David R. MacIver, et al., *Hypothesis*.

Considerations - Design by Contract[7]

```
function ret = sortbackwards( v );
  ttest.expect_isa( v, 'vector' );  % precondition
  POSTCONDITION( @(ret) numel(ret)==numel(v) );  % postc.
  ret = sort( v, 'descend' );
```

- Preconditions
- Postconditions *(partly experimental)*
- (Class invariants)

[7] Bertrand Meyer, *Applying "Design by contract"*, 1992.

Considerations - Design by Contract[7]

```
function ret = sortbackwards( v );
  ttest.expect_isa( v, 'vector' );  % precondition
  POSTCONDITION( @(ret) numel(ret)==numel(v) );  % postc.
  ret = sort( v, 'descend' );
```

- Preconditions
- Postconditions *(partly experimental)*
- (Class invariants)

[7] Bertrand Meyer, *Applying "Design by contract"*, 1992.

Considerations - Design by Contract[7]

```
function ret = sortbackwards( v );
  ttest.expect_isa( v, 'vector' );  % precondition
  POSTCONDITION( @(ret) numel(ret)==numel(v) );  % postc.
  ret = sort( v, 'descend' );
```

- Preconditions
- Postconditions *(partly experimental)*
- (Class invariants)

[7] Bertrand Meyer, *Applying "Design by contract"*, 1992.

Considerations - Utilities - Executing Subfunctions

- Testing private functions: Necessary in Matlab

```
>> allfunctionhandle( 'spy' );
{@defaultspy}    {@iterapp}    {@iterchk}
{@itermsg}       {@nestdiss}   {@colamdmex}
{@symamdmex}
```

(This is actually not possible in Matlab)

Refactoring

Considerations - Refactoring

Refactoring: Part of the *TDD* cycle.

- `assignat`
- `captureat`
- `flowat`
- `inputat`
- `evalat`
- (`errorat`)

The Proof

The Proof (Now even on Octave)

Theorem

$$2 = 3$$

The Proof (Now even on Octave)

Theorem

$$2 = 3$$

Proof.

- ```
  evalat( 'in','EXPECT_EQ', ...
          'at',1, ...
          'eval',['if( isequal(varargin{1},2) );' ...
                  '  varargin{1}=3; end;'        ] );
  ```
  ```

The Proof (Now even on Octave)

Theorem

$$2 = 3$$

Proof.

- `evalat( 'in','EXPECT_EQ', ...`
  `'at',1, ...`
  `'eval',['if( isequal(varargin{1},2) );' ...`
  `'  varargin{1}=3; end;' ] );`

- `EXPECT_EQ( 2, 3 )`

The Proof (Now even on Octave)

Theorem

$$2 = 3$$

Proof.

- ```
  evalat( 'in','EXPECT_EQ', ...
          'at',1, ...
          'eval',['if( isequal(varargin{1},2) );' ...
                  '  varargin{1}=3; end;'       ] );
  ```

- ```
  EXPECT_EQ( 2, 3 )

  ans =
    logical
     1
  ```

□

Hacks

Boring hacks

Hacks - Performance

- Assertions: mostly stand alone
- Sections: overloading

Hacks - Performance

- Assertions: mostly stand alone
- Sections: overloading

Hacks - Gold standard tests

```
classdef CACHE < handle
 properties
  c;  % content
 end
 methods
  function obj = CACHE(c_)
   try;
    st = load('cache.mat');
    obj.c = st.x;
   catch me;
    obj.c = c_; end;
  end
  function set.c(obj, c_)
   obj.c = c_;
   x = obj.c;
   save('cache.mat', 'x');
  end
 end
end
```

```
x = CACHE( 'Hello' )
  % x.c = 'Hello'
x.c = 'Bye'
clear x
y = CACHE( 'Hello' )
  % y.c == 'Bye'
```

Interesting Hacks
Code injection hacks

Hacks - Code injection

- Conditional breakpoints[8]
- always return a falsy[9] value: wrap code
- Only evalable strings: store function handles
- Exceptions
- Octave

[8] Per Isakson, *tracer4m*, 2016.

[9] A value which implicitly converts to `false`.

Hacks - Code injection

- Conditional breakpoints[8]
- always return a falsy[9] value: wrap code
- Only evalable strings: store function handles
- Exceptions
- Octave

[8] Per Isakson, *tracer4m*, 2016.

[9] A value which implicitly converts to `false`.

Hacks - Code injection

- Conditional breakpoints[8]
- always return a falsy[9] value: wrap code
- Only evalable strings: store function handles
- Exceptions
- Octave

[8] Per Isakson, *tracer4m*, 2016.

[9] A value which implicitly converts to `false`.

Hacks - Code injection

- Conditional breakpoints[8]
- always return a falsy[9] value: wrap code
- Only evalable strings: store function handles
- Exceptions
- Octave

[8] Per Isakson, *tracer4m*, 2016.

[9] A value which implicitly converts to `false`.

Hacks - Code injection

- Conditional breakpoints[8]
- always return a falsy[9] value: wrap code
- Only evalable strings: store function handles
- Exceptions
- Octave

[8] Per Isakson, *tracer4m*, 2016.

[9] A value which implicitly converts to `false`.

Hacks - Code injection - evalat

```
function ret = evalat( fun, lne, h );
  persistent cache;
  if( nargin==0 );
    ret = cache;
    return; end;
  cache = h;
  h = ['returnfalse( ' ...
       '  assign( ''ttest_handle'', evalat() ) ) ||' ...
       'returnfalse( ttest_handle() );                '];
  dbstop( 'in',fun, 'at',num2str(lne), 'if',h );
```

```
function ret = returnfalse( varargin )
  ret = false;

function ret = assign( nme, val );
  assignin( 'caller', nme, val );
```

Hacks - Code injection - Early return

```
function ret = errorat( fun, lne )
  if( nargin==0 );
    evalin( 'caller', 'clear' );
    ret = false;
    return; end;
  dbstop( 'in',fun, 'at',num2str(lne), ...
          'if','errorat()' ); end;
```

```
>> errorat( 'spy', 42 );
>> spy;
Reference to a cleared variable marker.
Error in spy (line 42)
if isempty(marker), marker = '.'; end
```

Hacks - Code injection - `allfunctionhandle`

```
function ret = allfunctionhandle( fun )
  persistent cache;
  if( iscell(fun) );
    cache = fun;
    evalin( 'caller', 'clear' );
    ret = false;
    return; end;
  dbstop( 'in',fun, 'at','1', ...
          'if','allfunctionhandle(localfunctions)' );
  try;
    eval( fun ); end;
  dbclear( 'in',fun );
  ret = cache; end;
```

Interesting Hacks
Strange Hack

Hacks - TRACING

- Passing arguments multiple layers deep in the call stack

```
function f()
 a = 1234;
 g();

function g()
 b = 4321;
 h()

function h()
 b = evalin( 'caller', 'b' );   % OK
 a   % ??
```

Hacks - TRACING

- Passing arguments multiple layers deep in the call stack

```
function f()
 a = 1234;
 g();

function g()
 b = 4321;
 h()

function h()
 b = evalin( 'caller', 'b' );  % OK
 a  % ??
```

```
function TRACE( g, a );  % idea only
 cmd = eval(['@(tr_' num2str(a) ') g()']);
 cmd();
```

- `cmd` is an anonymous function with signature `@(tr_1234) g()`

→ Can be retrieved via `dbstack`

```
TTEST( 'tellme' )
```

TTEST('tellme') - Outlook

- more generators for PBT (GIVEN)
- more benchmark tests (MINTIME, MAXTIME)
- conditionally executable sections (TESTCASE, SECTION)
- code coverage reports
- more utilities for testing calculations with floating point numbers (ALMOST_EQUAL, RANGE, NEAR)
- other people using it

`TTEST( 'tellme' )` - Miscellanea

- Source:
  - *gitlab.com/tommsch/ttest* (Down until December)
  - *gitlab.com/TTEST_ICST/ttest* (Temporary until December)
- Installation
  - Download
  - run `TTEST install`
- Documentation
- Usages:
  - *Auditory Modelling Toolbox*
  - *Large Time Frequency Analysis Toolbox*
  - *ttoolboxes* (JSR and subdivision theory)
  - *TTEST*
- Licence: *Mozilla Public License 2.0*

`TTEST( 'tellme' )` - Miscellanea

- Source:
  - *gitlab.com/tommsch/ttest* (Down until December)
  - *gitlab.com/TTEST_ICST/ttest* (Temporary until December)
- Installation
  - Download
  - run `TTEST install`
- Documentation
- Usages:
  - *Auditory Modelling Toolbox*
  - *Large Time Frequency Analysis Toolbox*
  - *ttoolboxes* (JSR and subdivision theory)
  - *TTEST*
- Licence: *Mozilla Public License 2.0*

`TTEST( 'tellme' )` - Miscellanea

- Source:
  - *gitlab.com/tommsch/ttest* (Down until December)
  - *gitlab.com/TTEST_ICST/ttest* (Temporary until December)
- Installation
  - Download
  - run `TTEST install`
- Documentation
- Usages:
  - *Auditory Modelling Toolbox*
  - *Large Time Frequency Analysis Toolbox*
  - *ttoolboxes* (JSR and subdivision theory)
  - *TTEST*
- Licence: *Mozilla Public License 2.0*

`TTEST( 'tellme' )` - Miscellanea

- Source:
  - *gitlab.com/tommsch/ttest* (Down until December)
  - *gitlab.com/TTEST_ICST/ttest* (Temporary until December)
- Installation
  - Download
  - run `TTEST install`
- Documentation
- Usages:
  - *Auditory Modelling Toolbox*
  - *Large Time Frequency Analysis Toolbox*
  - *ttoolboxes* (JSR and subdivision theory)
  - *TTEST*
- Licence: *Mozilla Public License 2.0*

`TTEST( 'tellme' )` - Miscellanea

- Source:
  - *gitlab.com/tommsch/ttest* (Down until December)
  - *gitlab.com/TTEST_ICST/ttest* (Temporary until December)
- Installation
  - Download
  - run `TTEST install`
- Documentation
- Usages:
  - *Auditory Modelling Toolbox*
  - *Large Time Frequency Analysis Toolbox*
  - *ttoolboxes* (JSR and subdivision theory)
  - *TTEST*
- Licence: *Mozilla Public License 2.0*

That‘s it

That‘s it - Take home message

- Use TDD for development
- Consider using *TTEST* for scientific software in Matlab
- Use *Git* and *Continuous Integration (CI)*
  (not discussed, but important → left for the reader)

That's it - *TTEST* - Design