TEMPO Workshop on Software Development

Michal Kvasnica

STU

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

Agenda

Yesterday

- version control systems and collaborative development
- Mercurial, Git, BitBucket, GitHub
- providing support

Today

- unit testing
- documentation
- dissemination

Agenda

Yesterday

- version control systems and collaborative development
- Mercurial, Git, BitBucket, GitHub
- providing support

Today

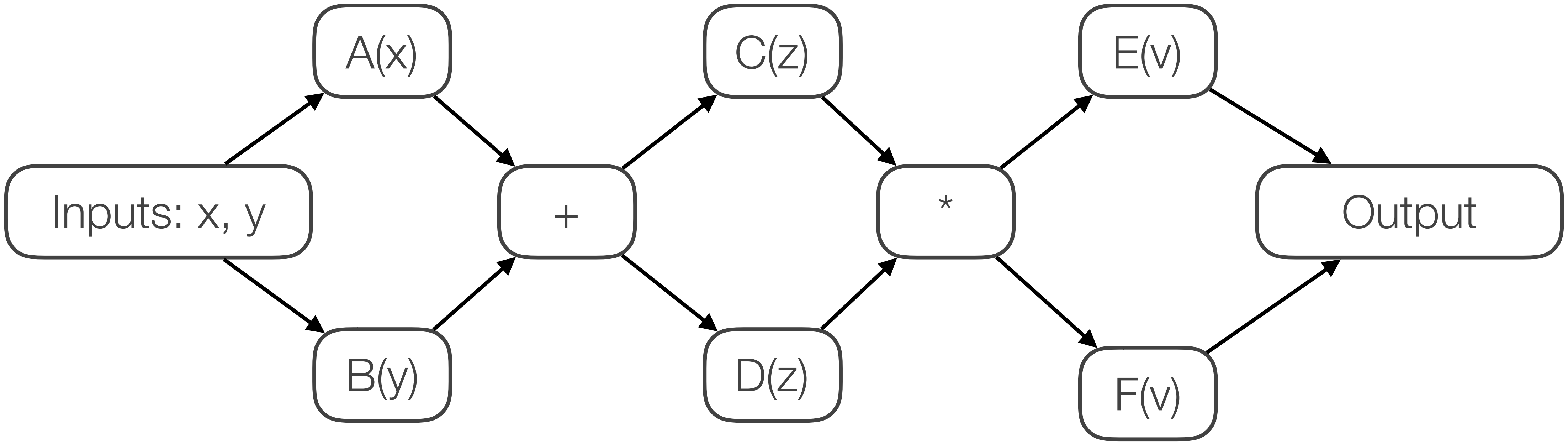
- **unit testing**
- documentation
- dissemination

BabyGuard™
INFANT SMART MAT
Your Baby's Breath Matters
World's First Baby Monitoring System that intelligently monitors your baby's breath movement count.
World's First Infant Smart Mat that Detects Subtle Movements
Validated in Singapore's Largest Baby Hospital
Ability to Detect Baby's Movement Pattern
Safe for the Baby
Peace of Mind
MODE
SELECT
BabyGuard
INFANT SMART MAT

Unit testing is like a baby breathing monitor for your code

Unit Testing

Imagine the output is incorrect. Where is the problem?



Solution: set of unit (atomic) tests for each function in your code

Purpose of unit testing:

- increase the confidence about the correctness of the code you write
- increase the confidence about the correctness of refactors
- make tracking down where a bug was introduced much simpler

Unit Testing: Example

Write a super-duper square root function `y = mysqrt(x)`

Before writing the code, specify the expected behavior:

```
function mysqrt_test1
assert(mysqrt(-1)==i);
assert(mysqrt(4)==2);
end
```

Unit Testing: Example

Write a super-duper square root function `y = mysqrt(x)`

Before writing the code, specify the expected behavior:

```
function mysqrt_test1
assert(mysqrt(-1)==i);
assert(mysqrt(4)==2);
end
```

Only now start writing the function

```
function y = mysqrt(x)
% Super-duper square root
y = x^0.5;
end
```

Unit Testing: Example

Write a super-duper square root function `y = mysqrt(x)`

Before writing the code, specify the expected behavior:

```
function mysqrt_test1
assert(mysqrt(-1)==i);
assert(mysqrt(4)==2);
end
```

Only now start writing the function

```
function y = mysqrt(x)
% Super-duper square root
y = x^0.5;
end
```

Execute your test suite often

- even after innocent-looking changes
- certainly before committting

```
e = 0.6;
y = x^e;
```

Unit Testing: Cover Edge Cases

When to add a unit test:

- before extending function's capabilities
- when a bug gets reported (each bug should translate into a unit test)

Example: `mysqrt([1 4 9])`

```
function y = mysqrt(x)
% Super-duper square root
y = x^0.5;
end
```

Unit Testing: Cover Edge Cases

When to add a unit test:

- before extending function's capabilities
- when a bug gets reported (each bug should translate into a unit test)

Example: `mysqrt([1 4 9])`

```
function y = mysqrt(x)
% Super-duper square root
y = x.^0.5;
end
```

Unit Testing: Cover Edge Cases

When to add a unit test:

- before extending function's capabilities
- when a bug gets reported (each bug should translate into a unit test)

Example: `mysqrt([1 4 9])`

```
function y = mysqrt(x)
% Super-duper square root
y = x.^0.5;
end
```

New unit tests:

```
function mysqrt_test2
% mysqrt with vectors
input = [-1 4 9];
expected = [i 2 3];
actual = mysqrt(input);
assert(isequal(actual, expected));
end
```

```
function mysqrt_test3
% mysqrt with matrices
input = [1 -1; 16 25];
expected = [1 i; 4 5];
actual = mysqrt(input);
assert(isequal(actual, expected));
end
```

Writing Good Unit Tests

A good unit test:

- does not depend on the environment and on other tests
- does not have side effects (e.g. modification of files)
- tests a single unit (method, function)
- tests edge cases
- provides a good coverage of the tested code
- runs fast (you will have hundreds of tests)
- is considered with the same value as the code (e.g. documentation)
- can be executed automatically

Writing Good Unit Tests

A good unit test:

- does not depend on the environment and on other tests
- does not have side effects (e.g. modification of files)
- tests a single unit (method, function)
- tests edge cases
- **provides a good coverage of the tested code**
- runs fast (you will have hundreds of tests)
- is considered with the same value as the code (e.g. documentation)
- can be executed automatically

Coverage Example

Super-duper square root function rejects strings:

```
function y = mysqrt(x)
% Super-duper square root
if ~isa(x, 'double')
    error('Only doubles please');
end
y = x.^0.5;
end
```

How many lines of `mysqrt.m` are executed by the unit test?

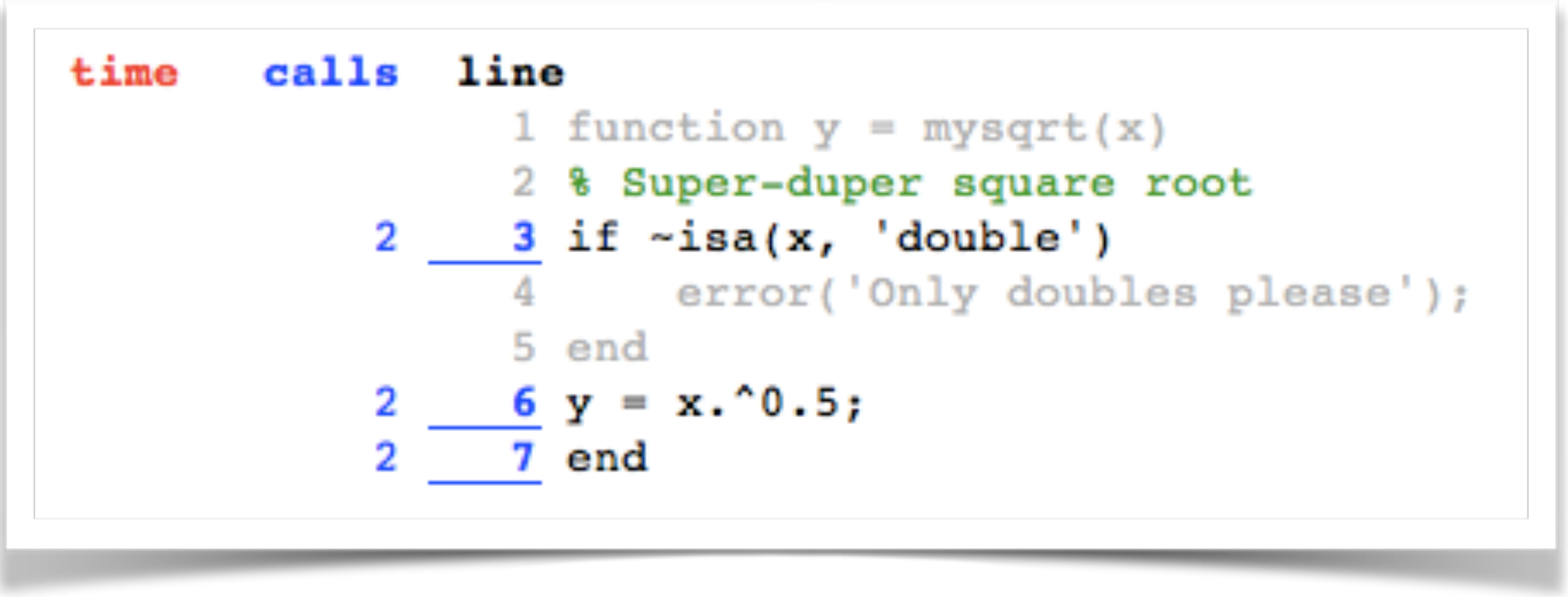
```
function mysqrt_test1
% mysqrt with scalars
assert(mysqrt(-1)==i);
assert(mysqrt(4)==2);
end
```

Coverage = (no. of lines executed)/(no. of lines total)*100%

Finding Coverage via Matlab Profiler

```
function y = mysqrt(x)
% Super-duper square root
if ~isa(x, 'double')
    error('Only doubles please');
end
y = x.^0.5;
end
```

```
function mysqrt_test1
% mysqrt with scalars

assert(mysqrt(-1)==i);
assert(mysqrt(4)==2);
end
```

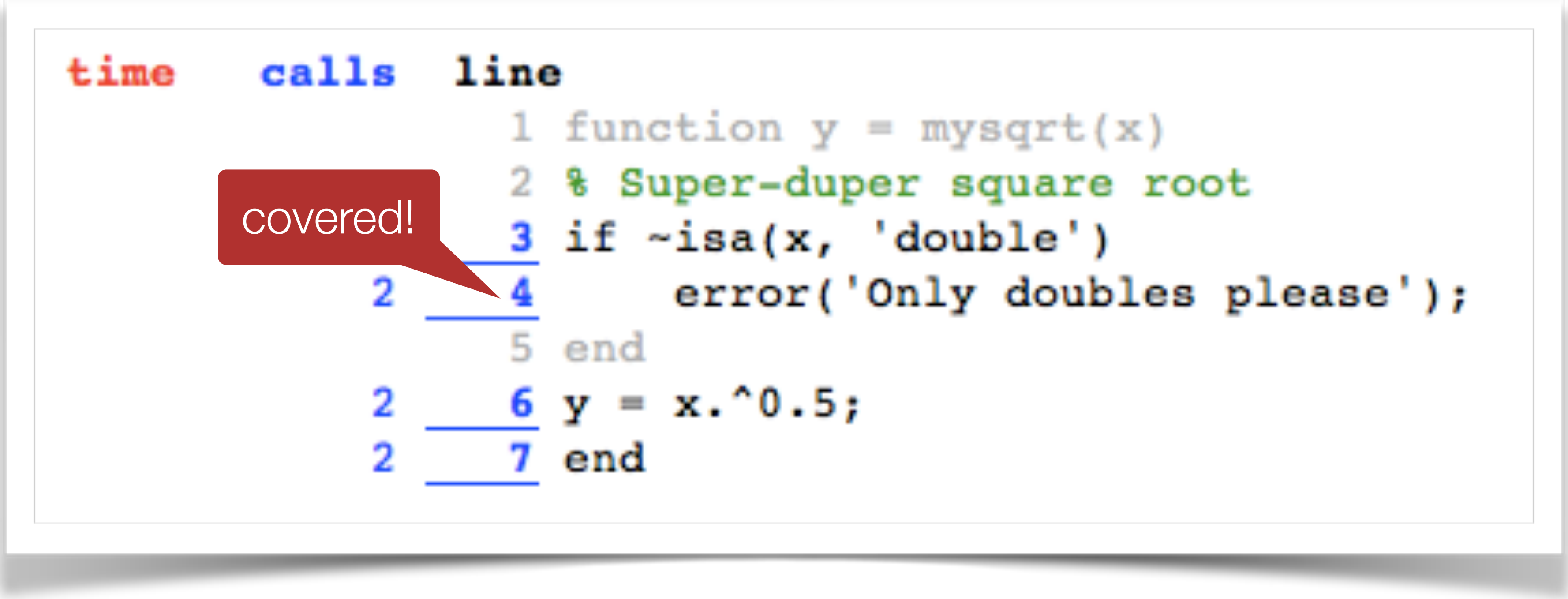


```
time    calls   line
                   1 function y = mysqrt(x)
                   2 % Super-duper square root
            2      3 if ~isa(x, 'double')
                   4     error('Only doubles please');
                   5 end
            2      6 y = x.^0.5;
            2      7 end
```

Finding Coverage via Matlab Profiler

```
function y = mysqrt(x)
% Super-duper square root
if ~isa(x, 'double')
    error('Only doubles please');
end
y = x.^0.5;
end
```

```
function mysqrt_test4
% mysqrt with non-doubles

assertError(@() mysqrt('hello'));
assertError(@() mysqrt(struct));
end
```



```
time    calls  line
                 1 function y = mysqrt(x)
                 2 % Super-duper square root
                 3 if ~isa(x, 'double')
           2     4     error('Only doubles please');
                 5 end
           2     6 y = x.^0.5;
           2     7 end
```

covered!

Writing Good Unit Tests

A good unit test:

- does not depend on the environment and on other tests
- does not have side effects (e.g. modification of files)
- tests a single unit (method, function)
- tests edge cases
- provides a good coverage of the tested code
- runs fast (you will have hundreds of tests)
- is considered with the same value as the code (e.g. documentation)
- **can be executed automatically**

Unit Testing Frameworks for Matlab

Home-made solutions

Matlab unit testing framework (since R2013a)

MOxUnit + MOcov

Unit Testing Frameworks for Matlab

Home-made solutions

Matlab unit testing framework (since R2013a)

MOxUnit + MOcov

Unit Testing in MPT

1838 tests as of November 2016

- implemented as Matlab functions organized in subdirectories
- one test often checks several edge cases
- basic test suite runs in 10 minutes

Home-made test runner: `run_all_mpt_tests`

- measures the runtime (some tests are meant for stress-testing)
- links errors to the editor
- allows to execute tests selectively
- lets to re-run failed tests

run_all_mpt_tests

Demo

Unit Testing Frameworks for Matlab

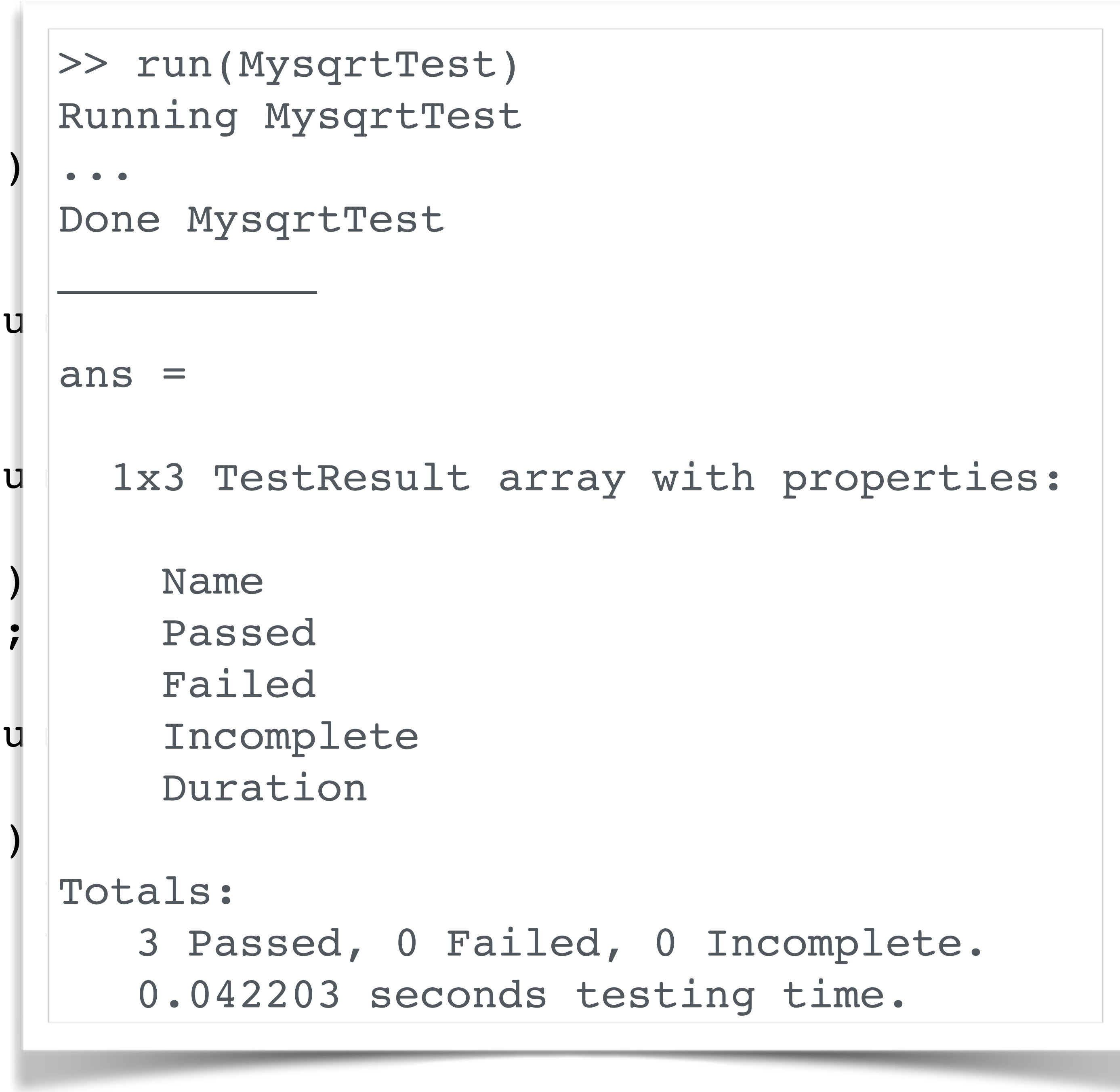
Home-made solutions

Matlab unit testing framework (since R2013a)

MOxUnit + MOcov

Matlab Unit Testing Framework

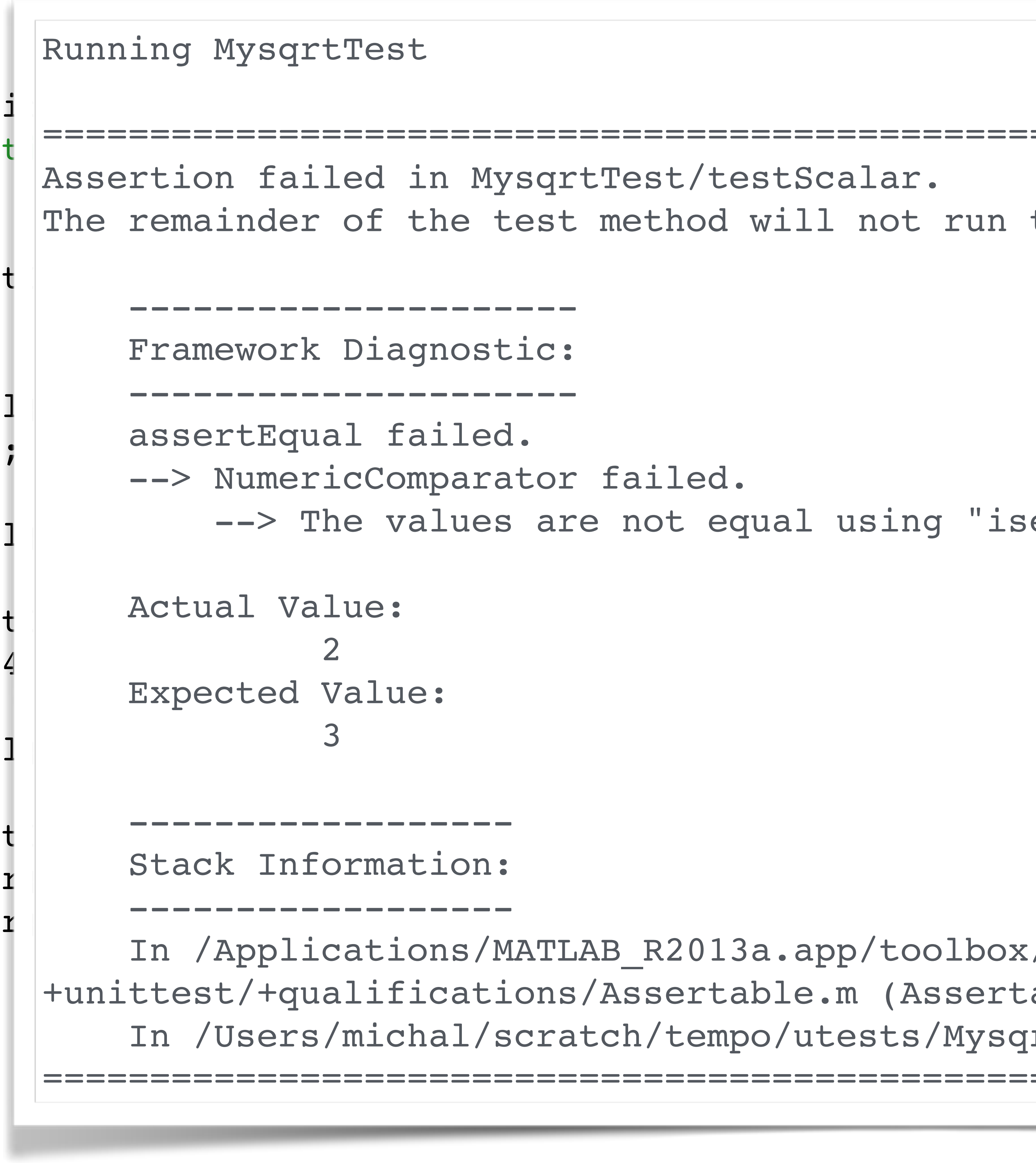
```
classdef MysqrtTest < matlab.unittest.TestCase
    % tests for the mysqrt function

    methods (Test)
        function testScalar(testCase)
            actual = mysqrt(4);
            expected = 2;
            testCase.assertEqual(actu
            actual = mysqrt(-1);
            expected = i;
            testCase.assertEqual(actu
        end
        function testVector(testCase)
            actual = mysqrt([1 4 9]);
            expected = [1 2 3];
            testCase.assertEqual(actu
        end
        function testDouble(testCase)
            testCase.assertError(@()
            testCase.assertError(@()
        end
    end

end
```



```
>> run(MysqrtTest)
Running MysqrtTest
...
Done MysqrtTest
__________


ans =


  1x3 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration

Totals:
   3 Passed, 0 Failed, 0 Incomplete.
   0.042203 seconds testing time.
```

Matlab Unit Testing Framework

```
classdef MysqrtTest < matlab.uni
    % tests for the mysqrt funct

    methods (Test)
        function testScalar(test
            actual = mysqrt(4);
            expected = 3;
            testCase.assertEqual
            actual = mysqrt(-1);
            expected = i;
            testCase.assertEqual
        end
        function testVector(test
            actual = mysqrt([1 4
            expected = [1 2 3];
            testCase.assertEqual
        end
        function testDouble(test
            testCase.assertError
            testCase.assertError
        end
    end

end
```



```
Running MysqrtTest

=========================================
Assertion failed in MysqrtTest/testScalar.
The remainder of the test method will not run t

    ----------------------
    Framework Diagnostic:
    ----------------------
    assertEqual failed.
    --> NumericComparator failed.
        --> The values are not equal using "ise

    Actual Value:
             2
    Expected Value:
             3

    -------------------
    Stack Information:
    -------------------
    In /Applications/MATLAB_R2013a.app/toolbox/
+unittest/+qualifications/Assertable.m (Asserta
    In /Users/michal/scratch/tempo/utests/Mysqr
=========================================
```

Sidenote: Checking Equality

In Matlab: 3*0.1 != 0.3

Always include a tolerance when checking equality

- `assertEqual(a, b, 'AbsTol', tol)` checks |a-b| ≤ tol
- `assertEqual(a, b, 'RelTol', tol)` checks |a-b| ≤ tol.*|b|

Matlab Unit Testing Framework

- Various qualifications (same for `assume`…, `verify`…)
    - `assertTrue(actual)`
    - `assertFalse(actual)`
    - `assertEqual(actual, expected, 'AbsTol', a, 'RelTol', r)`
    - `assertNotEqual(actual, notExpected)`
    - `assertEmpty(actual)`
    - `assertSize(actual, expectedSize)`
    - `assertSubstring(actual, substring)`
    - `assertError(@() function, identifier)`
    - `assertWarning(@() function, identifier)`
    - `assertWarningFree(@() function)`
    - `...`

Matlab Unit Testing Framework

```
classdef MysqrtTest < matlab.unittest.TestCase
    % tests for the mysqrt function

    methods (Test)
        function testAssert(testCase)
            testCase.assumeEqual(actual, expected);
            testCase.verifyEqual(actual, expected);
            testCase.assertEqual(actual, expected);
            fprintf('End of testAssert.\n');
        end
   end
end
```

Different consequences when the statement is false:

- `assumeEqual`: abort the test, mark it as incomplete
- `verifyEqual`: continue with next line, mark the test as failed
- `assertEqual`: abort the test, mark it as failed

Matlab Unit Testing Framework

Demo

Agenda

Yesterday

- version control systems and collaborative development
- Mercurial, Git, BitBucket, GitHub
- providing support

Today

- unit testing
- **documentation**
- dissemination

Documentation

A good project includes:

- `README.md` file in the root of your repository
- inline help
- demos / examples
- static user guide: `mkdocs` + `readthedocs.org`
- dynamic user guide: `jupyter` + `mkdocs` + `readthedocs.org`

Documentation

A good project includes:

- **`README.md` file in the root of your repository**
- inline help
- demos / examples
- static user guide: `mkdocs` + `readthedocs.org`
- dynamic user guide: `jupyter` + `mkdocs` + `readthedocs.org`

README.md

Displayed when the repo is visited on Bitbucket/GitHub

Provides basic information:

- purpose of the tool
- prerequisites (e.g. Matlab)
- installation instructions
- links to additional resources (documentation, wiki, etc.)
- contact information (email, discussion group, issue tracker)
- license (usually in `LICENSE.md`, more on this later)

Use the markdown syntax

Demo

Documentation

A good project includes:

- `README.md` file in the root of your repository
- **inline help**
- demos / examples
- static user guide: `mkdocs` + `readthedocs.org`
- dynamic user guide: `jupyter` + `mkdocs` + `readthedocs.org`

Inline Help: Approach #1

Detailed help descriptions:

```
mpt_sysStructInfo Returns information about system structure

 [nx,nu,ny,ndyn,nbool,ubool,intInfo] = mpt_sysStructInfo(sysStruct)

 ---------------------------------------------------------------------------
 DESCRIPTION
 ---------------------------------------------------------------------------
 Returns number of states, inputs, outputs and number of dynamics contained in
 a given system structure.

 ---------------------------------------------------------------------------
 INPUT
 ---------------------------------------------------------------------------
 sysStruct  - system structure describing an LTI system

 ---------------------------------------------------------------------------
 OUTPUT
 ---------------------------------------------------------------------------
 nx        - number of states
 nu        - number of control inputs
 ny        - number of outputs
 ndyn      - number of dynamics
 nbool     - number of boolean inputs
 ubool     - indexes of integer (or boolean) inputs
 intInfo  - structure with information about overlapping dynamics
```

Inline Help: Approach #2

Matlab-based help descriptions:

```
function y = mysqrt(x)
% Super-duper square root
%
% y=mysqrt(x) computes the square root of X.
%
% X must be a double (scalar, vector, matrix).
if ~isa(x, 'double')
```

Preferred, help is not a substitute for a detailed user guide

```
end
```

```
>> help mysqrt
  Super-duper square root

  y=mysqrt(x) computes the square root of X.

  X must be a double (scalar, vector, matrix).
```

Documentation

A good project includes:

- `README.md` file in the root of your repository
- inline help
- **demos / examples**
- static user guide: `mkdocs` + `readthedocs.org`
- dynamic user guide: `jupyter` + `mkdocs` + `readthedocs.org`

Demos / Examples

Provide commented Matlab code that can be directly executed

```
% compute the square root of a scalar
y = mysqrt(5);
fprintf('The square root of 5 is: %f\n', y);

% plot the square root over an interval
x = linspace(0, 10, 100);
y = mysqrt(x);
plot(x, y);
```

Demos / Examples

Provide commented Matlab code that can be directly executed

```
% compute the square root of a scalar
y = mysqrt(5);
fprintf('The square root of 5 is: %f\n', y);

% plot the square root over an interval
x = linspace(0, 10, 100);
y = mysqrt(x);
plot(x, y);
```

Typically demos are the first thing users execute

- use scripts, not functions
- keep them simple to understand!
- make them fast to execute (no expensive computations)

Consider the demos as basic unit tests

Documentation

A good project includes:

- `README.md` file in the root of your repository
- inline help
- demos / examples
- **static user guide: `mkdocs` + `readthedocs.org`**
- dynamic user guide: `jupyter` + `mkdocs` + `readthedocs.org`

`mkdocs`

Python-based static site generator (`pip install mkdocs`)

Uses the markdown syntax with support for:

- LaTeX expressions
- tables
- figures

Live reload on save

Markdown files can be directly viewed on Bitbucket/GitHub

mkdocs

Start a new documentation (creates `docs/` and `mkdocs.yml`)

```
$ cd youproject
$ mkdocs new .
```

Serve the documentation locally

```
$ mkdocs serve
$ open http://127.0.0.1:8000
```

Edit files (browser will automatically reload)

```
$ edit mkdocs.yml
$ edit docs/index.md
$ add new .md files to docs/
```

Build&deploy HTML versions if necessary (creates `site/`)

```
$ mkdocs build
```

Demo

readthedocs.org

Free cloud hosting for your documentation

Tied to your public repository

Automatically builds HTML docs from markdown sources after each new commit

Demo

Documentation

A good project includes:

- `README.md` file in the root of your repository
- inline help
- demos / examples
- static user guide: `mkdocs` + `readthedocs.org`
- **dynamic user guide: `jupyter` + `mkdocs` + `readthedocs.org`**

Jupyter Notebooks

Command line on steroids

Fusion of markdown-styled comments, code, and results

Supports almost any language: python, Julia, Matlab, ...

Can export to HTML, PDF, markdown (for integration with `mkdocs`)

Jupyter Notebooks

Demo

Agenda

Yesterday

- version control systems and collaborative development
- Mercurial, Git, BitBucket, GitHub
- providing support

Today

- unit testing
- documentation
- **dissemination**

Dissemination

Getting your code into the hands of users involves:

- licensing
- packaging
- distribution / installing / updating

Dissemination

Getting your code into the hands of users involves:

- **licensing**
- packaging
- distribution / installing / updating

Licensing

Open-source licenses

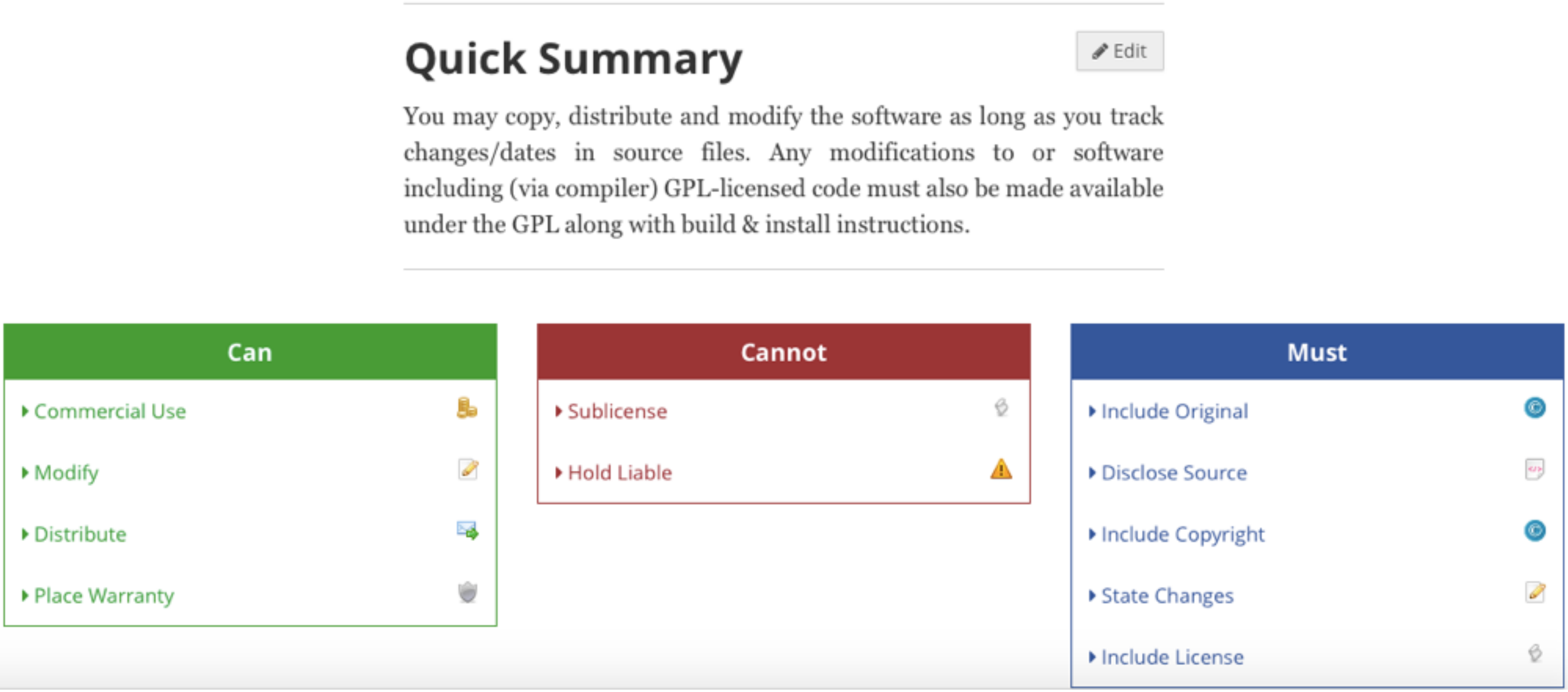
- GPL2, GPL3, MIT, BSD 2-clause, BSD 3-clause, Apache, …

Semi-open licenses

- YALMIP

Closed/commercial licenses

tldrlegal.com



Quick Summary

You may copy, distribute and modify the software as long as you track changes/dates in source files. Any modifications to or software including (via compiler) GPL-licensed code must also be made available under the GPL along with build & install instructions.

Can	Cannot	Must
Commercial Use	Sublicense	Include Original
Modify	Hold Liable	Disclose Source
Distribute		Include Copyright
Place Warranty		State Changes
		Include License

Comparison of Licenses

	Commercial use	Sublicense	Hold liable	Modify	Disclose source	Include copyright	Include license
GPL	✔	✗	✗	✔	!	!	!
MIT	✔	✔	✗	✔		!	!
BSD	✔		✗	✔		!	!

✔ can ✗ cannot ! must

https://tldrlegal.com

The YALMIP License

Copyright owned by Johan Löfberg

YALMIP must be referenced when used in a published work
(give me some credit for saving your valuable time!)

YALMIP, or forks or versions of YALMIP, may not be re-distributed as a part of a commercial product unless agreed upon with the copyright owner (if you make money from YALMIP, let me in first!)

YALMIP is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY, without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE
(if your satellite crash or you fail your Phd due to a bug in YALMIP, your loss!)

Forks or versions of YALMIP must include, and follow, this license in any distribution.

Licensing

My recommendations:

- YALMIP-style license if you care about recognition
- GPL license if you are an open-source fan/fanatic
- MIT license for practically oriented authors
  (allows sublicensing = making money from tailoring/consulting)

Always include `LICENSE.md` in the root of your repository

Dissemination

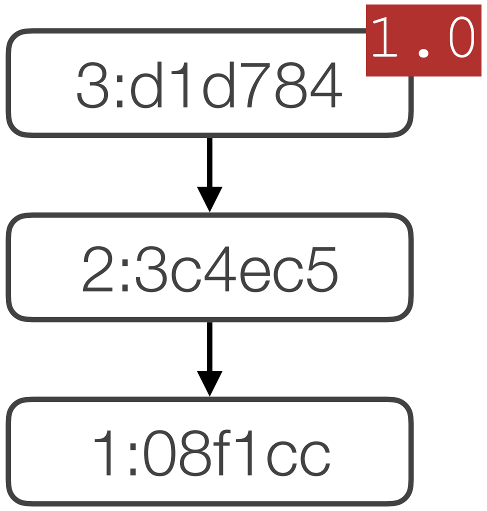
Getting your code into the hands of users involves:

- licensing
- **packaging**
- distribution / installing / updating

Packaging

Prepare for a release:

- make sure all tests pass
- bump the version number
- update release notes
- tag the version (`hg/git tag x.y.z`)
- build documentation



Packaging

Prepare for a release:

- make sure all tests pass
- bump the version number
- update release notes
- tag the version (`hg/git tag x.y.z`)
- build documentation

Create the installation package:

- do a clean checkout from the VCS
- remove debugging/testing files
- zip everything and upload to your server (including the docs)

Update the project's web page

Ideally, have an automated build script (or use `tbxmanager`)

Dissemination

Getting your code into the hands of users involves:

- licensing
- packaging
- **distribution / installing / updating**

Typical Life Cycle from Users' Perspective

First installation:

- download
- unzip
- set path

Updating:

- download
- unzip
- unset path to the old version
- set path to the new version
- delete old version

Bottom line: doing this manually is cumbersome

Better Solution: `tbxmanager`

`tbxmanager` is to Matlab what `apt-get` is to Linux:

- for end users: easily install, update, and uninstall Matlab packages
- for developers: easily disseminate packages & track usage

Available at `www.tbxmanager.com`

Open, anyone can register anything
(but we do not host download packages)

140 000+ packages installed since 2013 (1 every 10 minutes)

Notable users: MPT, YALMIP, OPTI Toolbox

`tbxmanager` – Web Interface

Demo

`tbxmanager` – Matlab Interface

Basic commands:

- list available packages: `tbxmanager show available`
- list installed packages: `tbxmanager show installed`
- install a new package: `tbxmanager install package_name`
- update all packages: `tbxmanager update`
- uninstall a package: `tbxmanager uninstall package_name`
- re-enable packages after restart: `tbxmanager restorepath`

Can abbreviate commands, e.g., `tbxmanager sh av`

More commands are available: `help tbxmanager`

`tbxmanager` – Matlab Interface

Demo

`tbxcli`

Automatic generation and upload of distribution packages

- configure a simple make script `tbxmake.m`
- execute `tbxmake`

Behind the scenes:

- zip is built
- zip is uploaded to your server
- a new version is created at `tbxmanager.com`

Demo

`tbxmanager` 2.0

New version is in progress (stalled since 2014)

- version control integration (Bitbucket, GitHub)
- better package discovery (tags, sorting, search)
- dependencies

Help needed!

- python server-side programming
- nicer web UI
- documentation

Available at `www.tbxmanager.com/v2/`

- just for testing purposes, not a production version!

Demo

Take-Home Messages

Use version control (really, it's a must nowadays)

Employ existing tools (don't spend (too much) time writing your own)

Do unit testing for peace of mind (and cover edge cases)

Write a good documentation (think from the users' perspective)

Automate as much as you can (package building, testing, dissemination)

Release early, release often (and don't be afraid of bugs)

Give great support to your users! (be responsive)

Feel free to contact me at `michal.kvasnica@stuba.sk`