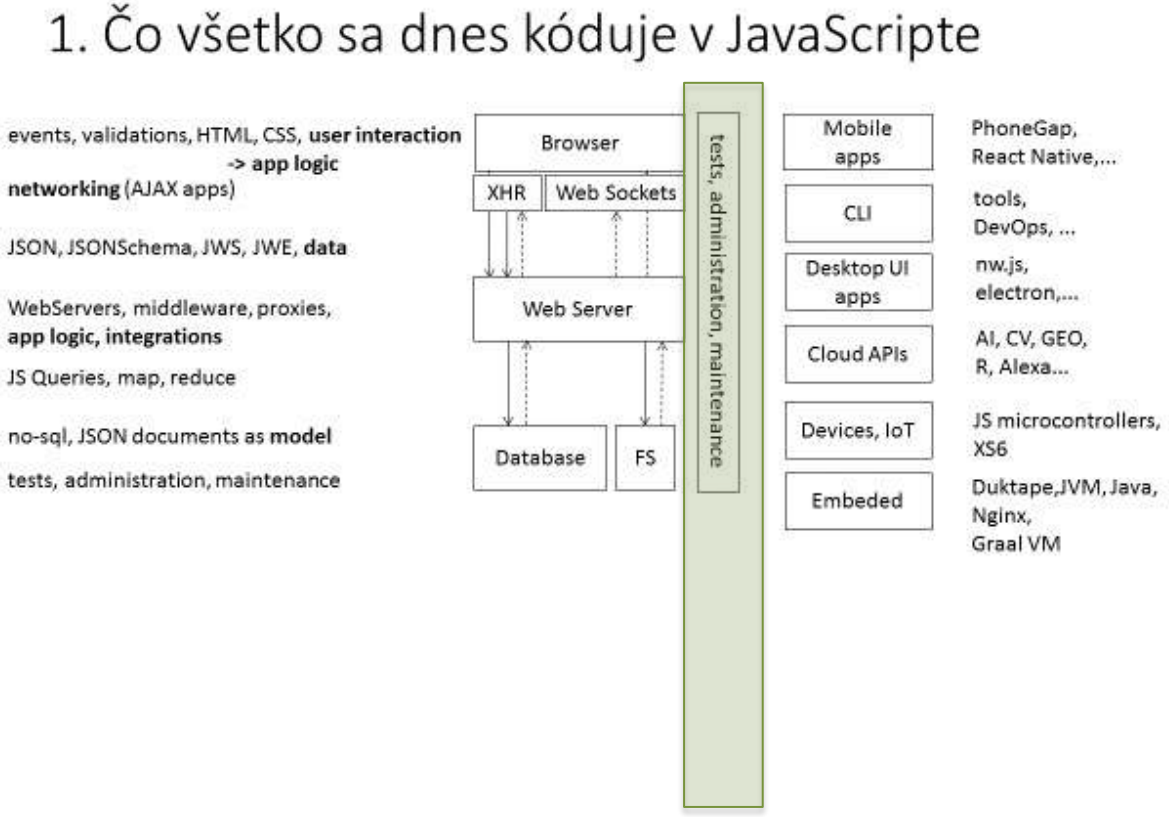
Testing

FIIT 2018

RECAP: Použitie JS

- Na prvej prednáške sme si hovorili, že okrem tvorby aplikácií rôzneho typuje je JS vhodný jazdyk na písanie testov a rôznych administratívnych a maintenance skriptov



Obsah

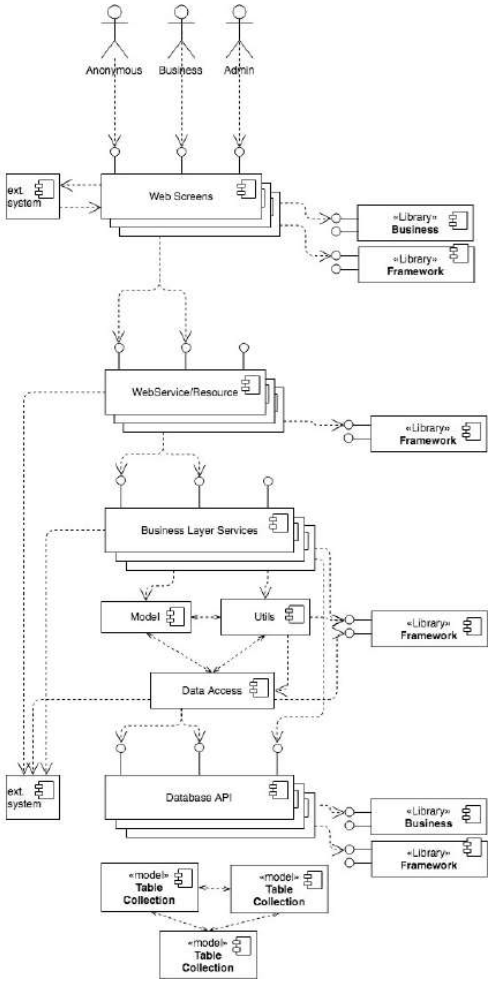
- Použitie JS na testovanie v reálnych projektoch, aj na veciach, ktoré nie sú napísané v JS
  - Typy testov v praxi
  - Ukážky JS kódov testov
- JS Test Frameworky
  - Porovnanie a stručné predstavenie
- Niektoré zaujímavé techniky
  - Code Coverage
  - Mutation Testing
- ...

Why Do We Need Tests ?

- To find bugs
- ??? there MUST be a better reasons
  - to describe and document features of application (BDD)
  - to map/learn application (Exploratory Testing)
  - to be able to refactor application (Unit Testing)
  - to be able to improve application (Performance, Configuration)
  - to ensure security of data, users and stability of system (Security Tests)
  - to be able to change parts and evaluate affected parts (Regression Testing)

- ....

Testing Classification (one of, simplified)



Scope/Level of testing (Which)

- Unit Test (component/interface test)
- Integration (internal, external)
- System

Objective of testing (What)

- Functional
- Usability (UX)
- Accessibility
- Internationalization
- Security
- Performance
- Load
- Stress
- Compatibility (XB, XC, XS)

Method of execution (how)

- Manual
- Automated
- Semi Automated

Time of test

- Alfa, beta, acceptance, regression,…

Knowledge

- White, black, gray box

Scenario

- Positive, negative

Degree

- Formal, exploratory, ad-hoc

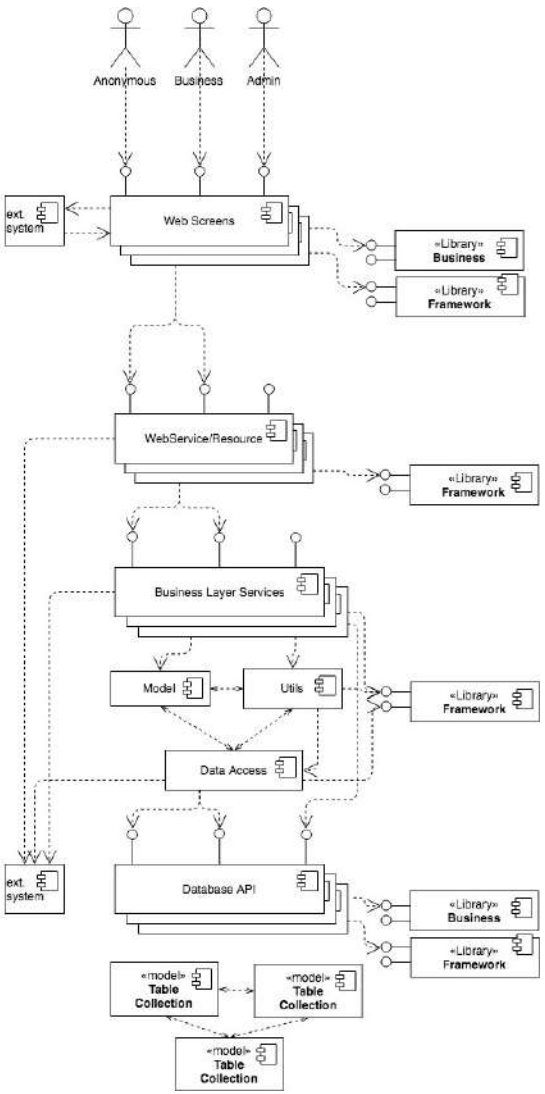
Other subtypes

- Mutation, code coverage, property, snapshot, configuration, exploratory testing

Methodologies

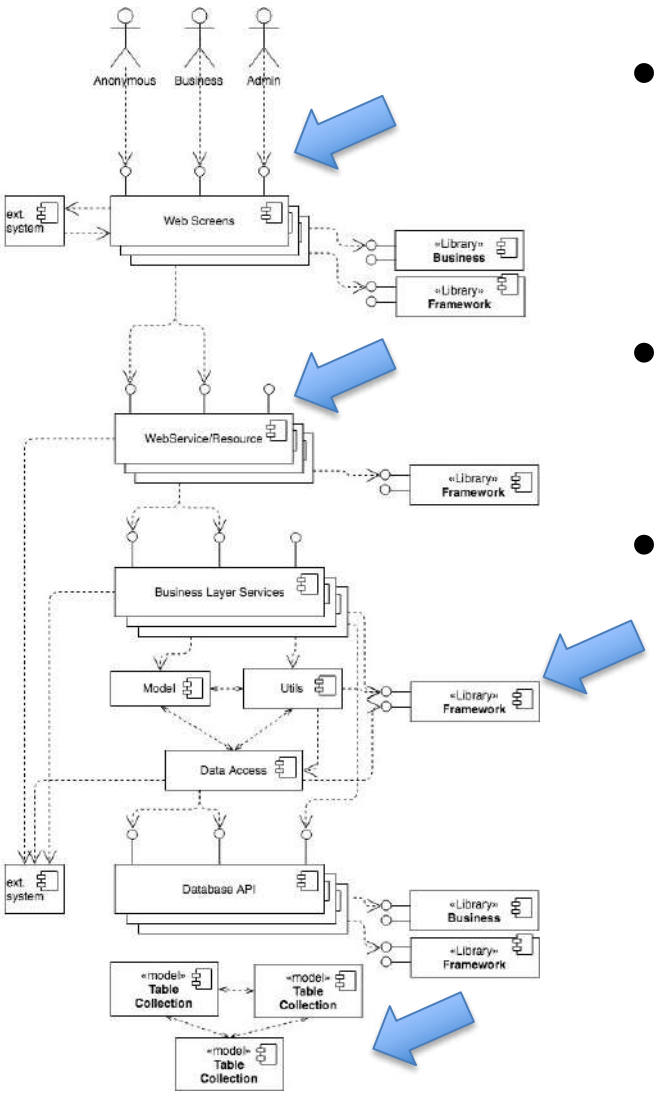
- TDD, ATDD, BDD

Testing Classification (super simplified)



- **Which** components/artifacts to test
  - Logical/physical architecture
  - UI, WS, BL, DB, libraries, infra, network, 2-together, all together
- **What** to test (objectives)
  - functionality
  - performance, security, ….
- **How** to test and evaluate
  - Manual
  - What tools, test APIs, test runners
  - When to run (commit, every build, smoke, once a month)
  - What approach (BDD, code coverage, snapshot, mutation,…)
  - Evaluation and reporting (clarity, formats, storage, trends, evaluation, base)

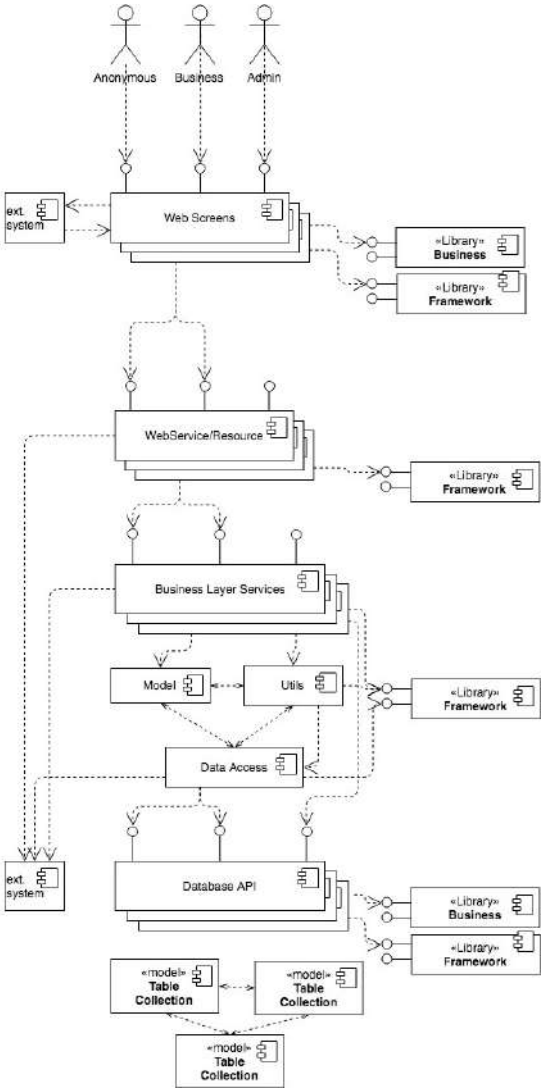
Practical testing



- **Using TDD** – all components have Unit Tests, most of them have integration tests
- **Using BDD** – strong functional tests, acceptance testing
- **Realita: Using XYZ** - tests are usually mixture of unit and integration tests, mixture of functional and non functional objectives, run ad-hoc or smoke

Tests and Reference Architecture

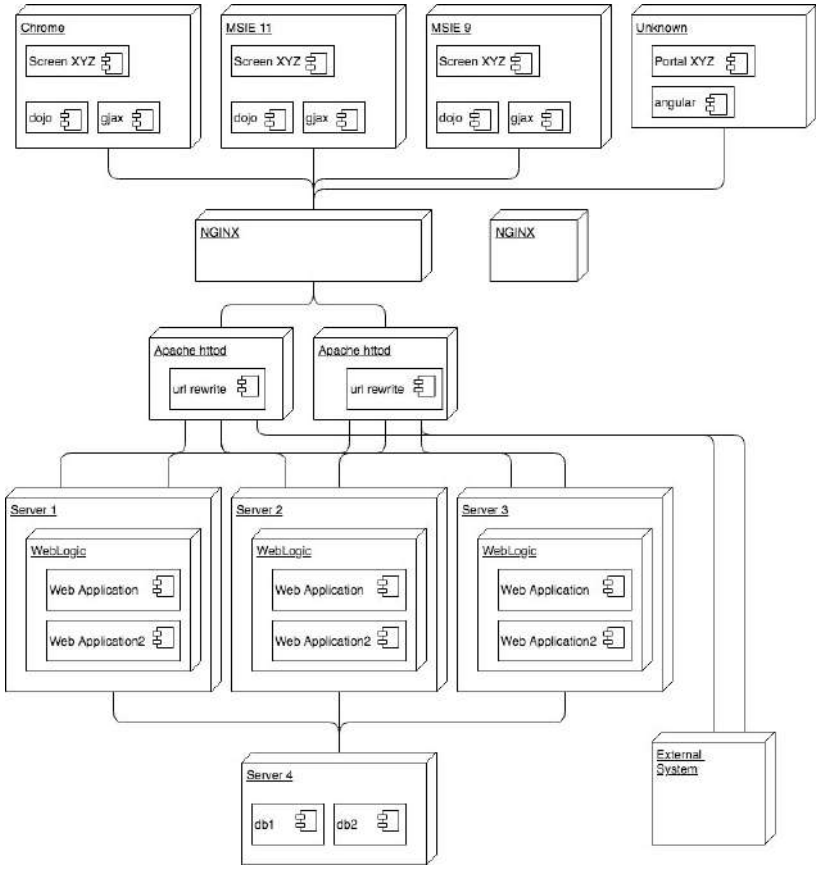
logical architecture



Layer	Test	Objective	Scope	Method	Time
UI					
	UI Functional Tests				
	UI Widget Tests				
	UI Libraries Unit Tests				
	Performance Tests				
	Accessibility Tests				
	XB Tests				
	Security Tests				
	UX testing				
Web Services					
	WS Forkflow Tests				
	WS Unit Tests				
	WS Performance Tests				
	WS Security Tests				
Business Layer					
	BL Services Tests				
	BL Components Test				
	Security Tests				
DB					
	DB API Tests				
	DB Unit Tests				
	DB Consistency Tests				
	DB API Perf Tests				
	DB Audit				

Tests and Reference Architecture

deployment

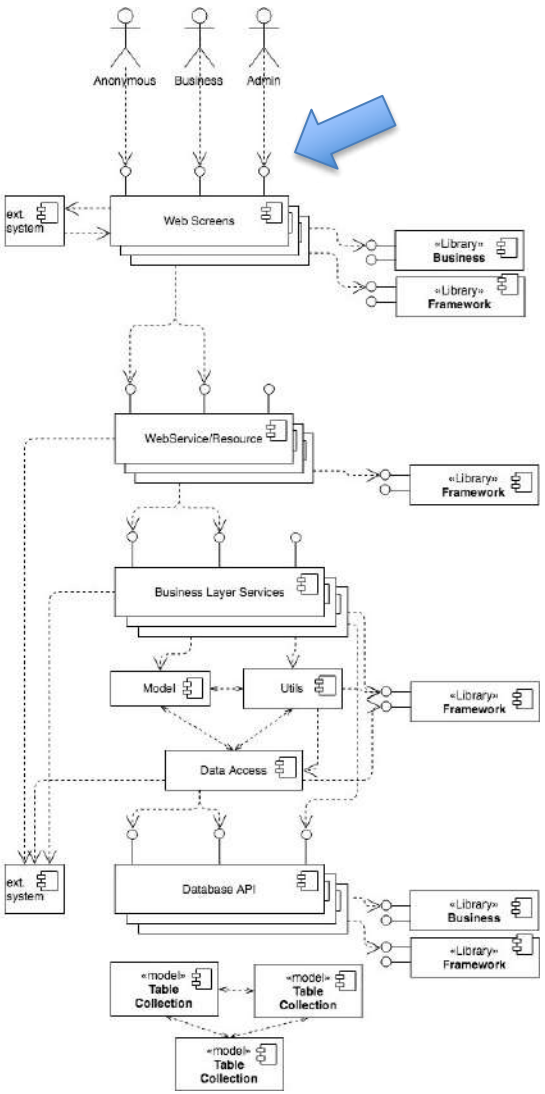


Test	Objective	Scope	Method	Time
HA/FO Tests				
Penetration Testing				
Performance Testing				
Compatibility Tests				
Configuration Testing				

Do We Need Tests ?

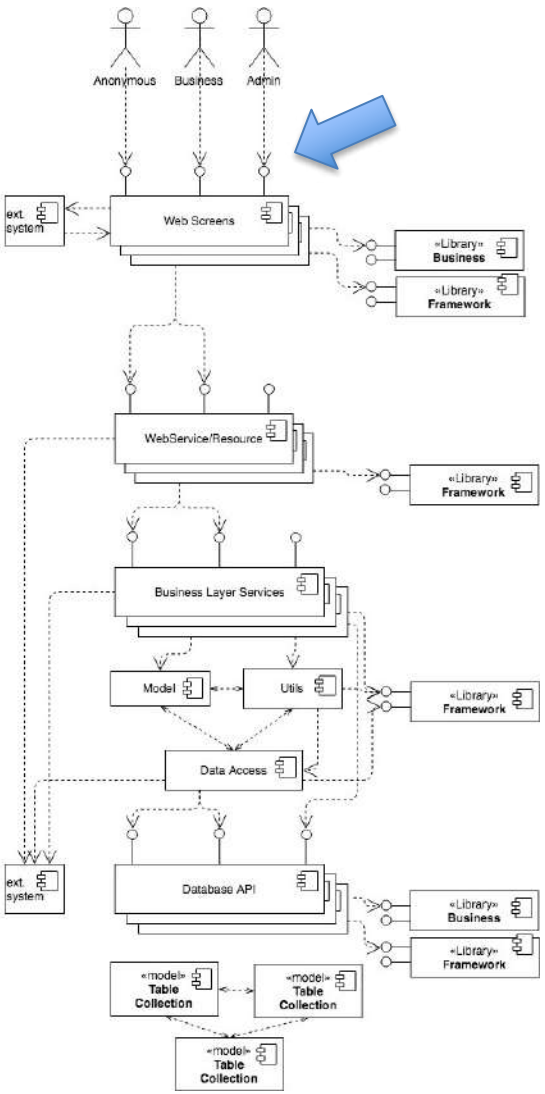
- Not always
- Ask the question **what bugs can be found** by what type of test
- Ask if there is better/easier/quicker/**cheaper** way to prevent bugs from happening
- Example with **test**:
  - UI calls WS, WS validates with JSON Schema,
  - schema changes because of CHR, UI is not adjusted
  - UI Test finds and reports bug (400 Bad Request)
  - UI is fixed
  - Test is run, green
- Example **without test:** alternative (automated review of changes):
  - detect all changes in schemas (git log)
  - detect all screens using given services using those schemas
  - review if screen have been adjusted as well (git log)
  - smoke test changed screens (manual)

UI Functional Tests



- **Objective**: functional
- **Method:** manual, automated, semi automated
- **Tools**: WebDriver API, Selenium, Katalon Studio, SikuliX
- **Scope**: screen, business processes
- **Time:** ad-hoc
- **Bugs Detected:**
  - UI <-----> WS communication protocol
  - Screen <-----> Screen
  - Screen States (enabled, disabled, readonly)
- **Problems and Challenges:**
  - selectors
- **Main Benefit:**
  - just prefill data on complex screens, rest to be done manually
  - generate screenshots and documentation

UI Functional Tests



Stats:

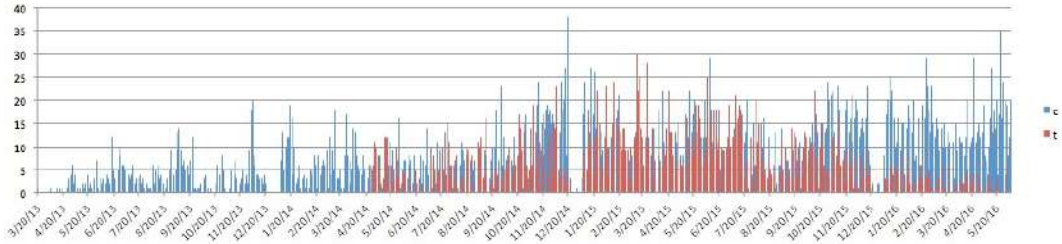
- Web Screens: **800**
- Unit Tests: **340**
- Screen Flow Tests: **61**

Effort:

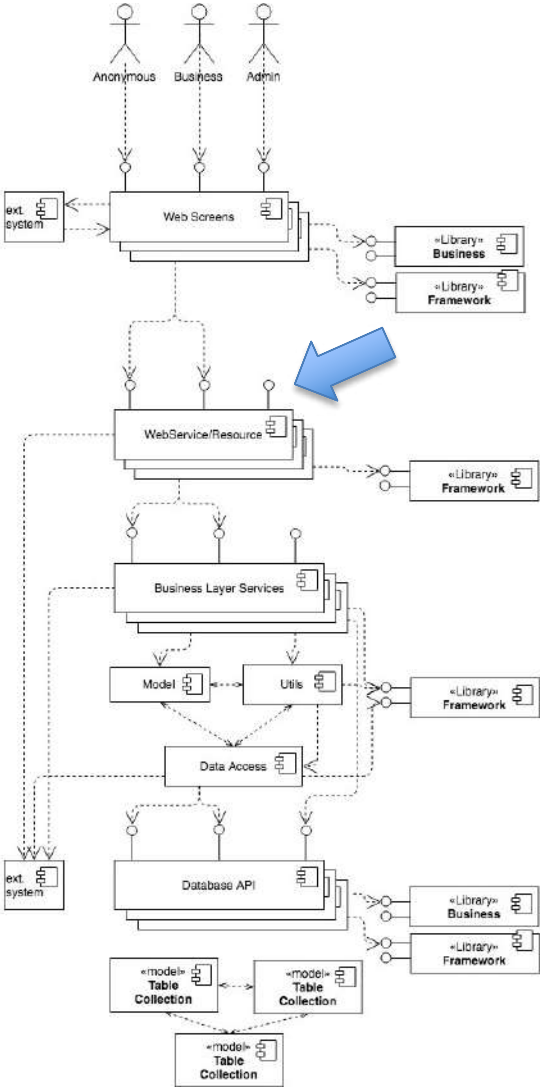
- JavaScript, 1000 files, **80000 LOC**
  - + 20 files, 2000 LOC in fmwk
- **3500** commits in 1 year, 2-3 developers

Outcome:

- Bugs Found: ???
- Other: **140 screenshots** from 70 screens
- Status: abandoned

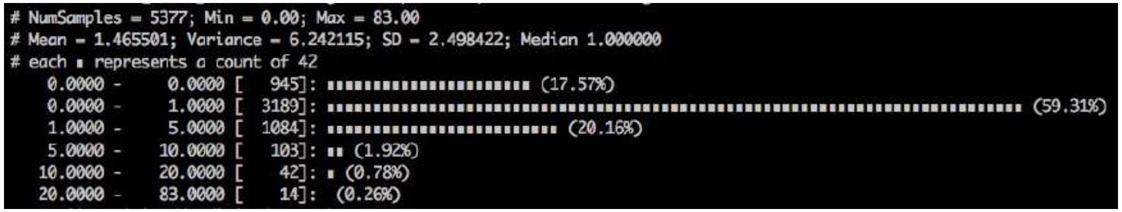


WS Unit Tests



Stats:

- Web Services: **5400**
- Unit Tests for: **4400** services (see below !)
- WS Workflow Tests: **???**



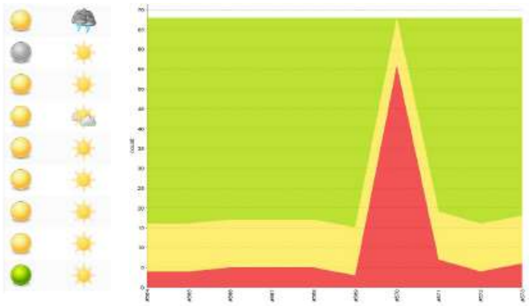
```
# NumSamples = 5377; Min = 0.00; Max = 83.00
# Mean = 1.465501; Variance = 6.242115; SD = 2.498422; Median 1.000000
# each ∎ represents a count of 42
    0.0000 -     0.0000 [   945]: ∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎ (17.57%)
    0.0000 -     1.0000 [  3189]: ∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎ (59.31%)
    1.0000 -     5.0000 [  1084]: ∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎ (20.16%)
    5.0000 -    10.0000 [   103]: ∎∎ (1.92%)
   10.0000 -    20.0000 [    42]: ∎ (0.78%)
   20.0000 -    83.0000 [    14]:  (0.26%)
```

Effort:

- Groovy, 4613 files, **177610 LOC**
  - **+ ??? test framework code**
- **9000** commits in 3 year, 10 – 15 developers

Outcome:

- Bugs Found: ???
- Other: ???
- Status: unstable

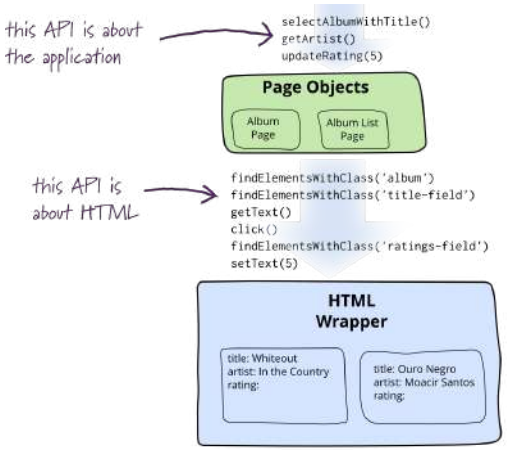


Test Code Examples

UI Functional Tests

Example:

- Tests also negative paths
- Written not recorded
- Mixture of various APIs:
  - Test harness API
  - Page Object
  - Web Driver API
- Label Based Selectors (our invention)
- Waiting For Elements to change
- sync/async mixture



https://martinfowler.com/bliki/PageObject.html

```
// ---------------------- tests ----------------------------
test.it("_navigate", function() {
    page.navigate();
    assert(api.input("Staré heslo").isVisible()).isTrue();
    assert(api.input("Nové heslo").isVisible()).isTrue();
    assert(api.input("Potvrď nové heslo").isVisible()).isTrue();

    reporter.save.all(this.test, "_navigate");
});
test.it("Submit blank", function() {
    page.navigate();
    api.button("Vykonaj zmenu").click();

    assert(api.input("Staré heslo").isInvalid()).isTrue();
    assert(api.input("Nové heslo").isInvalid()).isTrue();
    assert(api.input("Potvrď nové heslo").isInvalid()).isTrue();
});
test.it("Submit non-matching new pwd", function() {
    page.navigate();
    api.button("Vykonaj zmenu").click();

    api.input("Staré heslo").setValue("fooBar");
    api.input("Nové heslo").setValue("aaa");
    api.input("Potvrď nové heslo").setValue("bbb");

    assert(api.input("Staré heslo").isInvalid()).isFalse();
    assert(api.input("Nové heslo").isInvalid()).isFalse();
    assert(api.input("Potvrď nové heslo").isInvalid()).isTrue();

    assert(assert(api.input("Potvrď nové heslo")._get("message"))
        .is.equalTo("Nové heslo a potvrdenie nového hesla nie sú identické!"));

});
test.it("Same new as old", function() {
    page.navigate();

    api.input("Staré heslo").setValue(cfg.UI_USER);
    api.input("Nové heslo").setValue(cfg.UI_PWD);
    api.input("Potvrď nové heslo").setValue(cfg.UI_PWD);

    assert(api.input("Staré heslo").isInvalid()).isFalse();
    assert(api.input("Nové heslo").isInvalid()).isTrue();
    assert(api.input("Potvrď nové heslo").isInvalid()).isFalse();
});
```

WS Unit Tests

Example:

- Tests also negative paths
- Mixture of various APIs:
  - Test harness API
  - **HTTP Requests**
  - **Application**
- Partial Patches
- async by nature
- …

```
49  it(`invalid, empty payload shell return 400`, function() {
50
51      return assertRequest("POST", "svc/portal/hhi/policy/", { auth: portalUser }, {}, {
52          400: (body) => {} // now returns 500
53      });
54  });
55  it(`minimal input defined by schema shell work and return 200`, function() {
56      // minimal payload changed
57      var input = {
58          policyStartDate: today
59      };
60      return assertRequest("POST", "svc/portal/hhi/policy/", { auth: portalUser }, input, {
61              // asserts about
62              200: (output) => {
63                  assert("policyNumber" in output, "system assigns policyNumber");
64              }
65          })
66          .then(({ body }) => body);
67  });
68  it(`policyStartDate in PAST shell FAIL for GEN_PORTAL user`, function() {
69      // minimal payload changed
70      var input = {
71          policyStartDate: yesterday
72      };
73      return assertRequest("POST", "svc/portal/hhi/policy/", { auth: portalUser }, input, {
74              // asserts about
75              403: {
76                  "code": "policy_start_date_not_valid",
77              }
78          })
79          .then(({ body }) => body);
80  });
```

https://martinfowler.com/bliki/PageObject.html

BL Component Test

```
var printf = require("printf");
// TODO: fractional inches (?:(\d+) (\d+\/\d+)(?:''|["]))
// will require more refactor
var re = /(?:(\d+\.?\d*)['']([\d+\.?\d*)(?:''|["]))|(?:(\d+\.?\d*)(?:''|["]))|(?:(\d+\.?\d*)['''])/g;

function ft2cm(ftString, format) {

    format || (format = "%(cm).2f");
    return ftString.replace(re, function(m, f, i, ii, ff) {
        var cm = ((f || ff || 0) / 0.032808 + (i || ii || 0) / 0.39370);
        return printf(format, {
            cm: cm,
            m: cm / 100,
            mm: cm * 10,
            ft: m
        });
    });
}
ft2cm.APPEND_TRANSLATED = "%(ft)s (%(cm).2f cm)";
ft2cm.APPEND_ORIGINAL = "%(cm).2f cm (%(ft)s)";
module.exports = ft2cm;

```

- example: **nconv** – inches to cm conversion with formatting
- code: 1 function , 20 LOC
- test: 25 functions/tcs , 100 LOC
- TDD

```
f84c3de > 3 years, 1 month ago refactor for easier extensions [Marko Martin]
14      7       src/ft2cm.js

82314e3 > 3 years, 1 month ago [FIX] JSHint errors [Marko Martin]
2       1       .jshintrc
1       1       src/ft2cm.js

23d6ac2 > 3 years, 2 months ago sample added [Martin Marko]
11      1       README.md
3       2       src/index.js

a0f52c2 > 3 years, 2 months ago quite supported as inch mark as well [Martin Marko]
1       1       src/ft2cm.js
8       0       test/test.js

cfc6572 > 3 years, 2 months ago unicde foot and inch marks now supported, not only apos [Martin Marko]
2       1       src/ft2cm.js
22      9       test/test.js

66aab51 > 3 years, 2 months ago [FIX] seems like working for most usecases I need now [Martin Marko]
4       4       src/ft2cm.js
24      0       test/test.js

116d294 > 3 years, 2 months ago new failing test added [Martin Marko]
1       1       src/ft2cm.js
8       1       test/test.js

4caa722 > 3 years, 2 months ago initial alg and test [Martin Marko]
2       119     README.md
2       2       package.json
8       0       src/ft2cm.js
3       1       src/index.js
5       5       test/test.js
```

```
describe("f2cm", function() {

    it("feet and inches", function() {
        assert.equal(ft2cm("6'1''"), 185.42);
    });
    it("feet", function() {
        assert.equal(ft2cm("1'"), 30.48);
    });
    it("inches", function() {
        assert.equal(ft2cm("1''"), 2.54);
    });
     it("inches (whitespaces not supported)", function() {
        assert.equal(ft2cm("1 ''"), "1 ''");
    });
    //....
    //....
    it("detect and replace in string", function() {
        assert.equal(ft2cm("I'm 5'2'' high"), "I'm 157.48 high");
    });
    it("detect and append in string", function() {
        // TODO: fix the api of the function
        assert.equal(ft2cm("I'm 5'2'' high", "%(ft)s (%(cm).2f cm)"),
    });
    it("multiple times", function() {
        assert.equal(ft2cm("6'1'' and 6'1''"), "185.42 and 185.42");
    });
    it("feet space inches", function() {
        assert.equal(ft2cm("6' 1''"), "182.88 2.54");
    });
    it("toFixed", function() {
        assert.equal(ft2cm("6'", "%(cm).0f"), "183");
    });
    it("add units", function() {
        assert.equal(ft2cm("6'", "%(cm).0fcm"), "183cm");
    });
    it("add units", function() {
        assert.equal(ft2cm("6'", "%(cm).2f cm"), "182.88 cm");
    });
    //.......
    it("feet - Foot and inch marks", function() {
        assert.equal(ft2cm("6’"), "182.88");
    });
    it("inches - Foot and inch marks", function() {
        assert.equal(ft2cm("1”"), "2.54");
    });
    it("feet and inches - Foot and inch marks", function() {
        assert.equal(ft2cm("6’1”"), 185.42);
```

DB Consistency Test

Example: “overpaid invoices”

- query for bad records
- assert and fail, if any exists

Test side effects:

- **document field constraints** (grep collection names and assert texts)

```
1  var assert = require("assert");
2  module.exports = {
3      query: (db) => {
4          return db.collection("fm.finance.financeentry")
5          .find({
6              entryType: 'INV',
7              entryState: 'CONF',
8              transactionType: 'PREM',
9              'amountOutstanding.value': { $lt: 0 }
10         }).toArray()
11     },
12     assert: (incorrect) => {
13         assert.equal(incorrect.length, 0,
14             'amountOutstanding.value must be >= 0 \
15             for "invoices" '
16         )
17     }
18 }
```

https://martinfowler.com/bliki/PageObject.html

JavaScript Test Frameworks

from NONE to ELEPHANTS

What is test framework

- Syntax for tests
- Asserts
- Runner
- Reports
- Extras

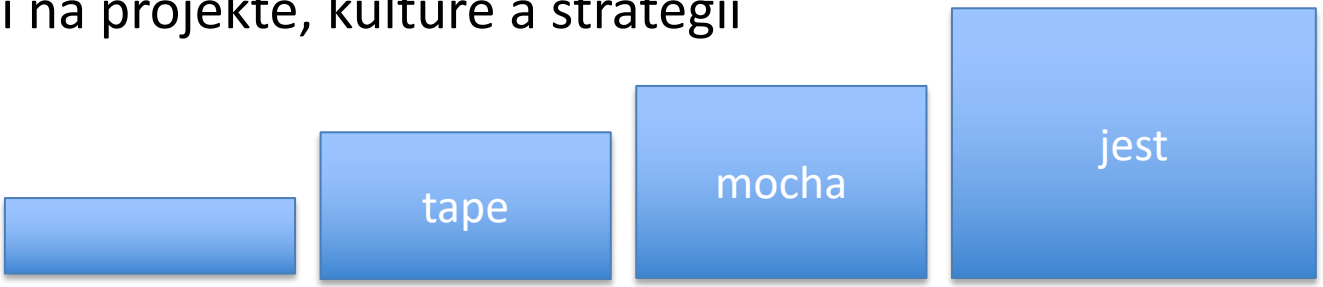
Špecifiká pre JS

- Izolácia (lebo všetko je mutable a máme global scope)
- Asynchronicita
- ....

- **Synax**: BDD, TDD, declarative, other....
- **Asserts**: basic, monitoring function calls (was called), fluent APIs, natural language
- **Runner**: node, bin, browser, parralel, async
- **Reports**: ansi, diff, json,html,tap,junit, ...
- **Extras**: code coverage, mutation testing, property testing, snashot testing,..

Test Frameworks Compared

- Frameworky navznikaly v najrôznejších podobách a veľkostiach
- Od minimalistických po kombinovateľné až po all-in-one
- Výber závisí na projekte, kultúre a stratégii



- Syntax for tests
- Asserts
- Runner
- Reports
- Extras

	tape	mocha	jest
	…	…	…
	…	…	…
		…	…
		…	…
		…	…
		…	…
		…	…
		…	…
		…	…
		…	…
		…	…
		…	…
		…	…
		…	snapshot tests

tape

```
$ npm ls --parseable | wc -l
      34
$ cloc node_modules/
     468 text files.
     434 unique files.
     105 files ignored.

github.com/AlDanial/cloc v 1.74  T=1.37 s (271.7 files/s, 20033.6 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JavaScript                     259           2464            801          14174
JSON                                                                       5216
Markdown                                                                   1885
YAML                                                                       1691
make                                                                         87
HTML                                                                         21
Bourne Shell                                                                  2
CoffeeScript                                                                  0
-------------------------------------------------------------------------------
SUM:                                                                      23076
-------------------------------------------------------------------------------
```

```
$ cloc node_modules/tape/
      85 text files.
      85 unique files.
       5 files ignored.

github.com/AlDanial/cloc v 1.74  T=0.23 s (353.6 files/s, 17610.5 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JavaScript                      76            465             79           3226
JSON                             1              0              0            153
YAML                             1              0              0             38
HTML                             1              0              0             21
Bourne Shell                     1              0              0              2
-------------------------------------------------------------------------------
SUM:                            80            465             79           3440
-------------------------------------------------------------------------------
```

mocha

```
$ npm ls --parseable | wc -l
      25
$ cloc node_modules/
     227 text files.
     217 unique files.
      41 files ignored.

github.com/AlDanial/cloc v 1.74  T=0.79 s (236.7 files/s, 57974.8 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JavaScript                     121           4714           7796          21716
Markdown                                                                  4483
JSON                                                                      4414
CSS                                                                        270
TypeScrip                                                                   58
YAML                                                                        53
make                                                                        40
HTML                                                                        18
----------------------------------------------------------------------------------
SUM:                                                                     31052
----------------------------------------------------------------------------------
```

```
$ cloc node_modules/mocha/
      49 text files.
      49 unique files.
       5 files ignored.

github.com/AlDanial/cloc v 1.74  T=0.24 s (183.7 files/s, 113176.6 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JavaScript                      39           2995           5849          13931
JSON                             1              0              0           1888
Markdown                         2            498              0           1606
CSS                              1             46             10            270
HTML                             1              0              0             18
-------------------------------------------------------------------------------
SUM:                            44           3539           5859          17713
-------------------------------------------------------------------------------
```

jest

```
$ npm list --parseable | wc -l
      582
$ cloc node_modules/
    8266 text files.
    6704 unique files.
Complex regular subexpression recursion limit (32766) exceeded at /Volumes/data/_WORK/init/node_modules/.bin/cloc line 9619.
    2034 files ignored.

github.com/AlDanial/cloc v 1.74  T=23.50 s (266.9 files/s, 32500.4 lines/s)
-------------------------------------------------------------------------------
Language                             files          blank        comment           code
-------------------------------------------------------------------------------
JavaScript                            4797          75244          82092         396884
JSON                                   698            541              0          96770
Markdown                               608          27686              2          64547
C/C++ Header                            21           1144            341           5869
XML                                     12            233             11           4481
YAML
TypeScript
make
HTML
Windows Module Defi
CSS
Python
C++
CoffeeScript
D
Handlebars
Bourne Again Shell
Bourne Shell
Lisp
DOS Batch
-------------------
SUM:
-------------------------------------------------------------------------------
```

```
$ cloc node_modules/jest-*
     217 text files.
     196 unique files.
      23 files ignored.

github.com/AlDanial/cloc v 1.74  T=0.82 s (235.8 files/s, 49459.8 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JavaScript                     153           4725           5881          24433
JSON                            28              0              0           3126
Markdown                        12            694              0           1742
TypeScript                       1             14              1             83
-------------------------------------------------------------------------------
SUM:                           194           5433           5882          29384
-------------------------------------------------------------------------------
```

Test Frameworks Compared

What's Wrong with Mocha, Jasmine, etc...?

1. **Too much configuration:** Choose an assertion library, chose a reporting library, chose a task runner (Grunt, Gulp, etc…) Then figure out how to translate the documentation examples to the reporting library / task runner you chose. All of this is too much cognitive load. *Vs: Choose Tape. Done.*

2. **Globals:** Mocha, Jasmine, and several other alternatives pollute the global environment with functions like `describe`, `it`, etc… Some assertion libraries extend built-in prototypes. Aside from removing the self-documenting nature of simple module exports, those decisions could potentially conflict with the code you're trying to test. *Vs: Tape's simple module export.*

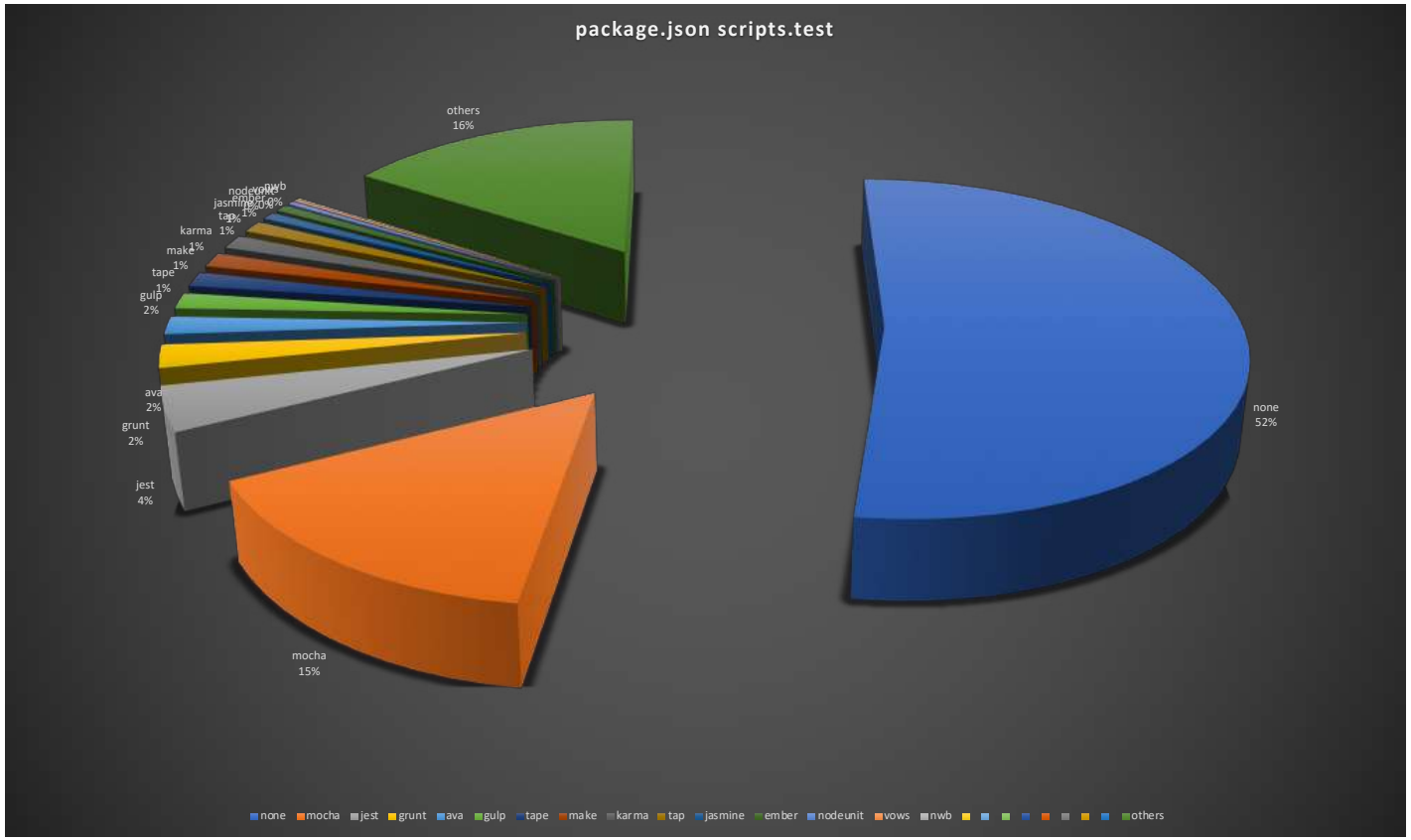
3. **Shared State:** Functions like `beforeEach` and `afterEach` actively encourage you to do something you **definitely should not do**: Share state between tests. *Vs. Tape: No such functions for global state sharing. Instead, call setup and teardown routines from individual tests, and* ***contain all state to local test variables.***

JEST and instance of not working because of Sandbox
https://github.com/facebook/jest/issues/2549

https://medium.com/javascript-scene/why-i-use-tape-instead-of-mocha-so-should-you-6aa105d8eaf4

Package.json#scripts.test

all	743102
none	406929
mocha	119403
jest	32534
grunt	16955
ava	13139
gulp	11849
tape	10884
make	10739
karma	10312
tap	8558
jasmine	5500
ember	5008
nodeunit	2396
vows	1378
nwb	941



From my **npma** project

MOCHA

mocha

- IMHO: Good **compromise** between minimal and maximal test framework
- Allows for several syntaxes of tests
- Assert library of choice
- Reporters, from standard TAP to parsable JSON, to human readable
- Integration with code coverage and other test tools (istanbul, striker, ...)
- Easy to learn

„*Test* ***Syntax*“ –** style of test DSL

Často sa používa fráza: test framework podporuje jednu zo syntaxí: BDD, TDD, xUnit, deklaratívne testy a podobne

- TDD a BDD sú development štýly a kľudne viete z BDD syntaxou napísať test pre TDD alebo z TDD syntaxou napísať UI test
- TDD produkuje testu viac zamerané na „ako je to implementované“
- BDD navádza na testy viac o testovaní “čo to robí“

Toto je trocha lepšia definícia:

Mocha’s “interface” system allows developers to choose their style of DSL. Mocha
has **BDD**, **TDD**, **Exports**, **QUnit** and **Require**-style interfaces.

BDD - style

Základ:

- describe(***what***)
- it(***shell do***)
- it(***shell not do***)

Nesting:

- context(***when***)
- context(***with***)

Setup:

- before(),
- after(),
- beforeEach(),
- afterEach()

```
describe('Array', function() {
  before(function() {
    // ...
  });

  describe('#indexOf()', function() {
    context('when not present', function() {
      it('should not throw an error', function() {
        (function() {
          [1,2,3].indexOf(4);
        }).should.not.throw();
      });
      it('should return -1', function() {
        [1,2,3].indexOf(4).should.equal(-1);
      });
    });
    context('when present', function() {
      it('should return the index where the element first appears in the array', function() {
        [1,2,3].indexOf(3).should.equal(2);
      });
    });
  });
});
```

TDD style

Základ:

- suite(),
- test()

Nesting:

- suite()

Setup:

- suiteSetup(), suiteTeardown(),
- setup(), and teardown()

```
suite('Array', function() {
  setup(function() {
    // ...
  });

  suite('#indexOf()', function() {
    test('should return -1 when not present', function() {
      assert.equal(-1, [1,2,3].indexOf(4));
    });
  });
});
```

Test Object style (exports syntax)

Základ:

- root {} (suite)
- properties (test)

Nesting:

- property nesting

Setup:

- keys before, after
- beforeEach, after Each

```
module.exports = {
  before: function() {
    // ...
  },

  'Array': {
    '#indexOf()': {
      'should return -1 when not present': function() {
        [1,2,3].indexOf(4).should.equal(-1);
      }
    }
  }
};
```

more declarative then previous styles,
but we are still coding tests, see DDT chapter

npx mocha **-ui=exports** ...

xUnit style (QUint style)

Základ:

- suite()
- test()

Nesting:

- order

Setup:

- before(), after(),
- beforeEach() and afterEach()

```
function ok(expr, msg) {
  if (!expr) throw new Error(msg);
}

suite('Array');

test('#length', function() {
  var arr = [1,2,3];
  ok(arr.length == 3);
});

test('#indexOf()', function() {
  var arr = [1,2,3];
  ok(arr.indexOf(1) == 0);
  ok(arr.indexOf(2) == 1);
  ok(arr.indexOf(3) == 2);
});

suite('String');

test('#length', function() {
  ok('foo'.length == 3);
});
```

require style

Základ:

- require().xyz

Načo:

- ak naozaj nechceme globálky z predošlých štýlov,
- alebo chceme vlastné názvy

```
var testCase = require('mocha').describe;
var pre = require('mocha').before;
var assertions = require('mocha').it;
var assert = require('chai').assert;

testCase('Array', function() {
  pre(function() {
    // ...
  });

  testCase('#indexOf()', function() {
    assertions('should return -1 when not present', function() {
      assert.equal([1,2,3].indexOf(4), -1);
    });
  });
});
```

Test code vs. Test Setup and Teardown

- Čo všetko mám napísať do it(), test() funkcie alebo niekam inam
- Kam inam ?
  - scope testu
  - before, after
  - beforeEach, afterEach
- Záleží:
  - v prvom priblížení má byť všetko v test()e, test má byť independent na akomkoľvek okolí
  - čo testujeme, ak benchamrky tak nechcem mať setup ako súčasť testu
  - ak nám ide o rýchlosť testu, asi nebudem opakovať vyhľadanie tovaru v databáze pre každý unit test
  - nezabudnúť na cleanup testu, zmazať temp fajly, pokazené záznamy v DB atď....

Test Setup and Teardown

should be used to:

- set up preconditions
  - navigate to tested page
  - insert test records
  - select records from DB
  - …
- clean up after your tests
  - close the page
  - delete test records
  - …

```
describe('hooks', function() {

  before(function() {
    // runs before all tests in this block
  });

  after(function() {
    // runs after all tests in this block
  });

  beforeEach(function() {
    // runs before each test in this block
  });

  afterEach(function() {
    // runs after each test in this block
  });

  // test cases
});
```

Passing values from test to test

- do not do this, each test is independent
- exception (accepted in my team) is to pass values from ***before***
- this can be done using ***scoped*** variable, or this context of suite/test

```
describe("PUT svc/user/${code}", function() {
    describe("duplicities", function() {
        var randomUser;
        before("search some available LDAP users", function() {
            return assertRequest("svc/groupInfo/users/?", {
                200: (body) => { assert(body.length > 0, "no unasigned LDAP users available ?") }
            }).then(({ body }) => assert(randomUser = random.item(body)));
        })

        it("insert duplicit user", function() {

            var code = randomUser.uid;
            var payload = {
                "blocked": false,
                "systemUser": false,
```

minimalistic testing

- https://medium.com/@WebReflection/vanilla-js-testing-part-ii-63b9d736121
- https://medium.com/@WebReflection/js-vanilla-test-code-coverage-7b7ba3740776

node.js codebase tests

how node.js team does tests

node.js codebase tests

- https://github.com/nodejs/node/blob/master/doc/guides/writing-tests.md

Assertion Libraries

none, simple, fluent, confusing, dangerous….

who writes the tests who reads the tests

Assertion Libraries

- Test Framework môže a nemusí obsahovať zabudovanú assert knižnicu
- Niektoré frameworky vám umožňujú knižnicu meniť, niektoré nie
- Assert knižnice môžete kombinovať, a podľa toho *aký test píšete*
- pre unit testy sú vhodnejšie tie *low level*
- pre funkčné a acceptance testy sú vhodnejšie tie *fluent a semantické*

- *Assert knižnice pre JS musia adresovať JS špecifické problémy*
  - *asynchronicitu (callbacks, Promises)*
  - *netypovosť*
  - *typy properties (own, enumerable,symbols)*
- *A problémy testovanej oblasti (biznis test vs. http, vs. stream test)*
  - *.....*
  - *assert.contains(arr, subarr)*
  - *assert.called(fn)*

Assertion Libraries - *style*

Tiež je zaužívaný pojem a členenie na TDD a BDD style podobne ako pri syntaxi testov

- BDD - expressive language & readable style, chainable
- TDD - clasical asserts

Samples:

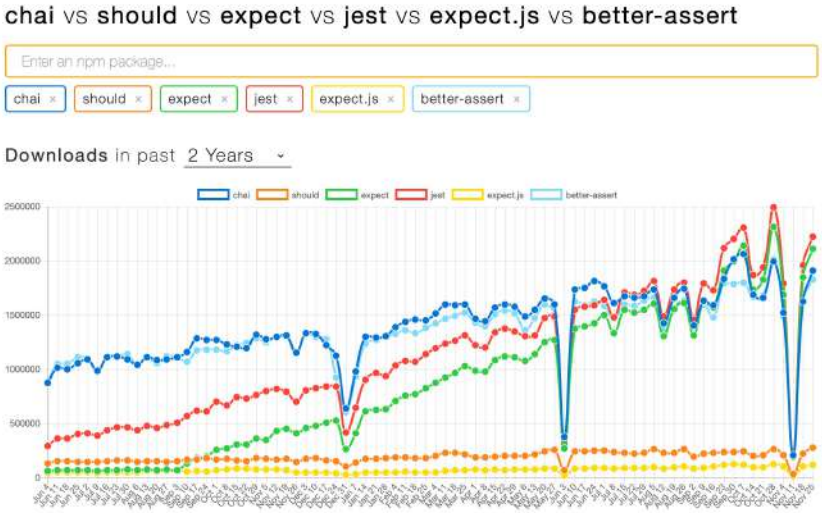
```
assert.typeOf(foo, 'string');
assert.equal(foo, 'bar');
assert.lengthOf(foo, 3)
assert.property(tea, 'flavors');
assert.lengthOf(tea.flavors, 3);
```

```
expect(foo).to.be.a('string');
expect(foo).to.equal('bar');
expect(foo).to.have.lengthOf(3);
expect(tea).to.have.property('flavors')
  .with.lengthOf(3);
```

```
foo.should.be.a('string');
foo.should.equal('bar');
foo.should.have.lengthOf(3);
tea.should.have.property('flavors')
  .with.lengthOf(3);
```

Assertion Libraries - npm

- **node assert module -** basic, simple, well defined, limited set of asserts
- **better-assert -** Better c-style assertions using callsite for self-documenting failure messages.
- **Chai -** BDD/TDD assertion library for node.js and the browser. Test framework agnostic**, *assert, expect, should styles***
- **Expect** - This package exports the expect function used in Jest. You can find its documentation on Jest's website.
- **Should -** should is an expressive, readable, framework-agnostic assertion library. The main goals of this library are to be expressive and to be helpful.



- *Expect.js* - not be confused with *expect*
- Expect is part of jest fmwk

- https://www.npmjs.com/package/chai
- https://jestjs.io/docs/en/expect.html
- https://jestjs.io/docs/en/expect.html#tohavereturned

How to choose Assertion Library

Factors to consider:

- exact documentation and stable implementation
  - je mi nanič assert ak neviem čo presne testuje, alebo ak sa definícia často mení, najme v *oblasti equal* a v oblasti *own, enumebrable, symbol* properties
- assertion capabilities
  - čo okrem základných assertov podporuje, ozaj ich potrebujem ? ci len kvoli reportingu ?
- assertion style
  - fluent, readable, chainable
- reporting quality
  - self describing, easy to detect error causes from messages
- extensibility
  - how easy can we extend, and combine asserts to create higher level assert
- ...
- ...

- migration and transferability – how well can you tests run on other test runners, ES versions, browsers, how much effort you need to migrate them if needed ?

node.js assert module

- Only ***TDD style - assert***.xyz(value, msg)
- **Well defined**/documented **equality rules**
- *coercing algorithms* are deprecated now

Only ***limited*** set of assertions covering 5 areas

- Truthy values
- Shallow equality
- Deep equality
- Throws not Throws Errors
- Promises

	API		Description
Truth	assert(value[, message])		The input that is checked for being **truthy**
	assert.ok(value[, message])		Tests if value is truthy. It is equivalent to assert.equal(!!value, true, message)
shallow equality	assert.equal(actual, expected[, message])	deprecated	Tests shallow, **coercive equality** using the **Abstract Equality Comparison (==).**
	assert.notEqual(actual, expected[, message])	deprecated	
	assert.strictEqual(actual, expected[, message])		Tests strict equality between the actual and expected parameters as determined by the **SameValue Comparison.**
	assert.notStrictEqual(actual, expected[, message])		
deep equality	assert.deepEqual(actual, expected[, message])	deprecated	Primitives, **Abstract Equality Comparison** (==). Objects: Only ***enumerable "own","non symbol"*** properties are considered
	assert.notDeepEqual(actual, expected[, message])	deprecated	
	assert.deepStrictEqual(actual, expected[, message])		Primitives: **SameValue Comparison**, Type Tags, Prototypes === , enumberable own properties **+ symbols**, unordered properties,...
	assert.notDeepStrictEqual(actual, expected[, message])		Tests for **deep strict inequality.** Opposite of assert.deepStrictEqual().
errors	assert.throws(fn[, error][, message])		Expects the function fn to throw an error.
	assert.doesNotThrow(fn[, error][, message])		Asserts that the function fn does not throw an error. is actually not useful
	assert.ifError(value)		testing the error argument in callbacks. **Throws value** if value is not undefined or null
	assert.fail([message])		throws an AssertionError with the provided error message or a default error message
	assert.fail(actual, expected[, message[, operator[, stackStartFn]]])	deprecated	
promises	assert.rejects(asyncFn[, error][, message])		Awaits the asyncFn promise or asyncFn() promise, check that the **promise is rejected**
	assert.doesNotReject(asyncFn[, error][, message])		Awaits the asyncFn promise or asyncFn() promise, check that the **promise is not rejected**

chai - 3 styles

A) assert – ala node.js (12) with some sugar (41/128), asserty na:

- na compare (lebo messages)
- chýbajú deepStrictEqual
- chýbajú promise asserty
- na typy (ozaj ich chceme ? ako ďalší rozmer chaosu)
- mutability
- object structures
- structural compare

Pozri: assert-libs.xlsx

https://www.chaijs.com/api/assert/

B) expect

- you chain together natural language assertions
- starts with **expect(something).**xyz.pqr()

C) should

- you chain together natural language assertions.
- starts with something.xyz.pqr()
- The should interface extends Object.prototype to provide a single getter as the starting point for your language assertions. It works on node.js and in all modern browsers except Internet Explorer.

What is really chai (expect style)

- https://www.chaijs.com/api/bdd/
- some examples are definitelly more readable then several asserts
- but are they more writable ? (learning, tools support, human errors)

- **41** - je v skutočnosti **assertov**
- **8** je ***modifikátorov***
  - .not,
  - .deep,.nested,.own,
  - .ordered,.any,.all,
  - .itself
- **16** timi noop ***slohovými*** slovíčkami
  - to,be,been,is,that,which,and,has,have,with,at,of,same,but,does,still
- Pozri: assert-libs.xlsx

chain expect style - explained

```
expect(beverages).to.have.property('tea').with.lengthOf(6);
```

start of chain
as function
not harmfull like
should style

garbage

assert

garbage

assert

in real it is:

```
expect(beverages)        .property('tea')     .lengthOf(6);
```

put any garbage in between, no one cares: allowed garbage is 16 words:

to,be,been,is,that,which,and,has,have,with,at,of,same,but,does,still

```
expect(beverages).of.is.but.has.property('tea').of.still.lengthOf(6);
```

chain expect style - explained

```
expect(beverages).    to.have    .property('tea');
```

garbage

assert

```
expect(beverages).not.to.have.own.property('tea');
```

modifier

modifier

in real it is:

```
expect(beverages).not          .own.property('tea');
```

chain *propper* **modifiers** to achieve more specized forms of asserts

.not,

.deep,.nested,.own,

.ordered,.any,.all,

.itself

assert libraries

- it is about all 3 things in this order:

1. SEMANTICS
2. REPORT hints
3. STYLE/SYNTAX

- nenadchýnajte sa syntaxou, kým nerozumiete semantike asertov

```
8   suite("Style comparison", function() {
9
10    let beverages = { tea: "Lipton" };
11    //let beverages = { tea: "Liption" };
12    //let beverages = {};
13    //let beverages = {tea:null};
14    //let beverages = {tea:[0,2,3,4,5,6]};
15    //let beverages = { tea: new Set([0, 2, 3, 4, 5, 6])};
16    //let beverages = Object.create({ tea: "Lipton" });
17    //let beverages = Object.defineProperty({}, "tea", { va
18
19    test("chai expect", () => {
20      let { expect } = require("chai");
21
22      expect(beverages).to.have.property('tea')
23        .with.lengthOf(6);
24
25    });
26    test("chai assert", () => {
27      let { assert } = require("chai");
28
29      assert.property(beverages, 'tea');
30      assert.lengthOf(beverages.tea, 6);
31
32    });
33    test("node assert", () => {
34      let assert = require("assert");
35
36      assert(beverages.hasOwnProperty("tea"));
37      assert.equal(beverages.tea.length, 6);
38
39    });
40    test("better-assert", () => {
41      let assert = require("better-assert");
42
43      assert(beverages.hasOwnProperty("tea"));
44      assert(beverages.tea.length === 6);
45
46    });
47  });
```

Assert Library Semantics - equals

- we have 4 algorithms for equal in JS
- we have no equal for structural equality
- we have no equal for deep equality

- how this maps to your testing library ?
- are you sure what you really test ?

- https://nodejs.org/api/assert.html#assert_assert_deepstrictequal_actual_expected_message
- https://github.com/chaijs/deep-eql
- ...

Assert Library Semantics - property

- own, enumerable,
  Symbols
- leads to other
  semantics of deepEqual,
  contains, includes etc…

Data Driven Tests

Data Driven Tests

- The simplest explanation of data-driven testing is this:
  - data that is external to your tests
  - not hardcoded
- What is data
  - inputs, expected outputs
  - test environment settings and control

Data Driven Tests

- Data can be externalized
  - JS Arrays and Objects and required, folders and files, XLSX table, database
- Main benefit is:
  - you can use “non testers” (business users) **to create test data**
  - **data can be transferred** from one architecture to another (run in JS on client, and in Java on Server)
  - you can use external test data for your implementation
  - …
  - D.R.Y logic of tests
  - but can be harder to quickly find out what the test is doing

Data Driven Tests in mocha

sample implementation

- test can be added dynamically [8,9] using mocha API
- find files/fixtures [6,7] and in test setup [10] add test for each file
- [16,20] – do whatever you want with the file inside test method samples/data-driven
- suite must have at least one static test for before() to execute [13]

```
1  const glob = require("glob-promise");
2  const assert = require("assert");
3
4  describe("Math.round DDT", function() {
5
6    const unroll = () =>
7      glob("./data/*", { cwd: __dirname });
8    const addTest = (file) =>
9      this.addTest(_it(file))
10   before("unroll", () => unroll().then(
11     files => files.forEach(addTest)
12   ));
13   it("unroll", () => assert(this.total() > 1));
14
15
16   function _it(test) {
17     return it(`testing: ${test}`, function() {
18       let { input, output } = require(test);
19       let actual = Math.round(input);
20       assert.equal(actual, output);
21     });
```

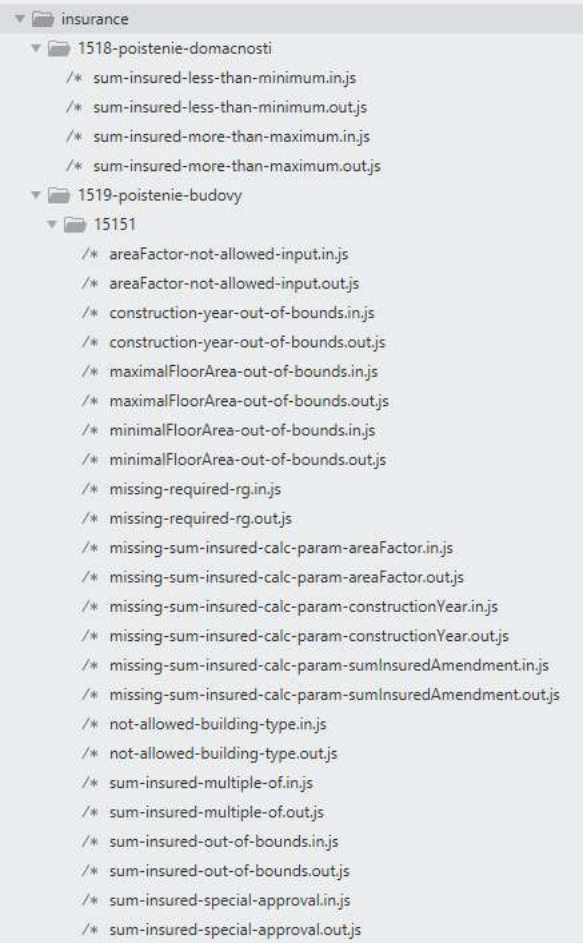
```
data-driven
  data
    test01.json
    test02.json
  round.spec.js
jest-project
```

test01.json | test02.json

```
1 {
2   "input": 10.2,
3   "output": 10
4 }
```

```
$ npx mocha --colors data-driven/round.spec.js

Math.round DDT
  √ unroll
  √ testing: ./data/test01.json
  √ testing: ./data/test02.json

3 passing (14ms)
```

Data as folders and files (example)



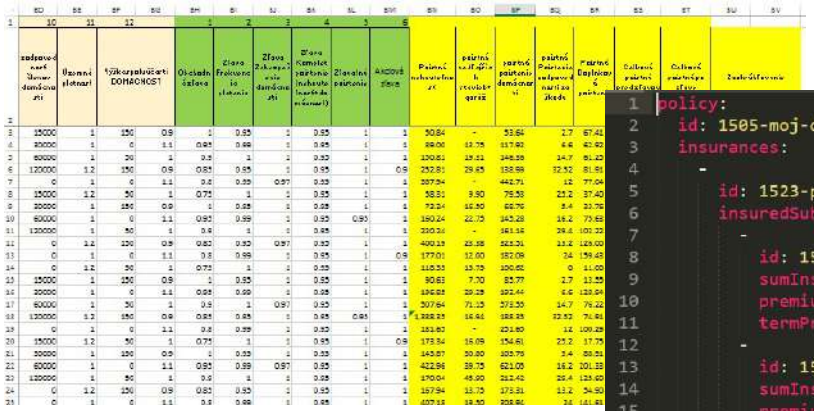
- JSON files organized under data/fixtures folder by convention: xyz.in, xyz.out
- generic runner loading files and generating describe()/it() hierarchy
- generic test code: assert.contains(alg(in),out)
- comparing only relevant parts (contains())
- format for expected errors

```
module.exports = {
    "policy": {
        "id": "1505-moj-domov-direct-byt",
        paymentFrequency: {
            id: "04-stvrtrocne",
            premiumInstallments: 4
        },
        "insurances": [{
            "id": "1523-poistenie-zodpovednosti",
            "insuredSubjects": [{
                "id": "15162-zodpovednost-domacnost",
                "sumInsured": 40000,
                "riskGroups": [{
                    "id": "01-zodpovednost"
                }]
            }, {
                "id": "15163-zodpovednost-budova",
                "sumInsured": 40000,
                "riskGroups": [{
                    "id": "01-zodpovednost"
                }]
            }]
        }]
    }
};
```

```
module.exports = {
    policy: {
        id: "1505-moj-domov-direct-byt",
        insurances: [{
            id: "1523-poistenie-zodpovednosti",
            insuredSubjects: [
                {
                    id: "15162-zodpovednost-domacnost",
                    sumInsured: 40000,
                    premium: 7.44,
                    termPremium: 1.86
                },
                {
                    id: "15163-zodpovednost-budova",
                    sumInsured: 40000,
                    premium: 2.16,
                    termPremium: 0.54
                }
            ],
            premium: 9.6,
            termPremium: 2.4
        }],
        premium: 9.6,
        termPremium: 2.4
    }
};
```

Formats of data

- technical:
  - Arrays, Objects, json
- non technical formats:
  - yaml
  - spreadsheet
  - DB with UI
- Organization may vary:
  - One file per input and one for output
  - One file per input and output
  - One file with many tests



```yaml
policy:
  id: 1505-moj-domov-direct-byt
  insurances:
    -
      id: 1523-poistenie-zodpovednosti
      insuredSubjects:
        -
          id: 15162-zodpovednost-domacnost
          sumInsured: 40000
          premium: 7.44
          termPremium: 1.86
        -
          id: 15163-zodpovednost-budova
          sumInsured: 40000
          premium: 2.16
          termPremium: 0.54
      premium: 9.6
      termPremium: 2.4
  premium: 9.6
  termPremium: 2.4
```

```json
{
    "name" : "path query and fragment use DIFFERENT ENCODE
    "expectedUri" : "/p/?q=q#f",
    "template" : "/${p}/?q=${q}#${f}",
    "param" : {
        "p" : "p",
        "q" : "q",
        "f" : "f"
    }
},
{
    "name" : "question mark is encoded in DIFFERENT way in path and fragment",
    "expectedUri" : "/%3F/?q=?#?",
    "template" : "/${p}/?q=${q}#${f}",
    "param" : {
        "p" : "?",
        "q" : "?",
        "f" : "?"
    }
},
```

```
// structure [inputs], output, "in"|"out"|any
// third parameter is used for post check
// - "in" - original input must equal current value of input (i.e. input was not modifed)
// - "out" - original input must equal return value (i.e. some inplace operation was done)
// - other - original input must equal other value (i.e. some inplace operation was done, but somethir
var testData = {
    "removeAll" : [
        [["a", "b", "c"],["b", "c" ]], ["a"], "in",
        [["a", "b", "c"],["b", "c", "a"]], [], "in",
        [["a", "b", "c"],["d", "e"]], ["a", "b", "c"], "in",
        [["a", "b", "c"],["b", "c", "d", "e"]], ["a"], "in"
    ],
    "retainAll" : [
        [["a", "b", "c"],["b", "c" ]], ["b", "c"], "in",
        [["a", "b", "c"],["b", "c", "a"]], ["a", "b", "c"], "in",
        [["a", "b", "c"],["d", "e"]], [], "in",
        [["a", "b", "c"],["b", "c", "d", "e"]], ["b", "c"], "in"
    ],
```

Snapshot Tests

Snapshot Tests

- Snapshot tests are a very useful tool whenever you want to make sure your code did not change unexpectedly.
- Písať testy na každú jednu property komplexných výstupov je problémové, testy už aj tak píšete na rôzne časti výstupu v rámci iných (unit) testov
- Cieľom testu je zistiť či sa výstup predošlého behu *zhoduje* z výstupom tohto behu testu
- Ak sa nezhoduje, reba došetriť či je to želaná zmena (pridané nejaké properties do objektu), alebo neželaná zmena (chuba kdesi v algoritme)
- Po došetrení buď nový výstup prehlásite za OK, alebo opravíte algoritmus

Snapshot Tests

- Princíp implementácie:
  - máte vstupy a k nim očakávane výstupy, kludne ponahrávané nejakým automatom prípadne vykonaním testu samotného
  - výstupy nekontrolujete a keď tak len zbezne, ručne, tie majú byť pokryté inými testami
- Musíte si niekde vstupy ukladať a verzionovat (disk a git)
- Musíte dokázať porovnať z predošlým snapshotom
- Musíte ho dokázať revertnúť alebo komitnúť (zase git)

Snapshot testing in mocha

sample implementation

- async unroll podobne ako v DDT
- nejaký spôsob načítania dát [19,20]
- vykonanie algoritmu nad vstupmi [22]
- nejaký assert na zhodnosť
- uloženie dát v prípade success a v prípade fail [26,28]
- fail testu v prípade failu assetu [29]

```
5  describe("Snapshot test", function() {
6
7    const unroll = () =>
8      glob("./data/*.in.*", { cwd: __dirname });
9    const addTest = (inputFile) => this.addTest(_it(
10     inputFile,
11     inputFile.replace(/[.]in[.](js|json)$/, ".out.json")
12   ))
13   before("unroll", () => unroll().then(files =>
14     files.forEach(addTest)));
15   it("unroll", () => {});
16
17   function _it(inputFile, outputFile) {
18     return it(`snapshot: ${inputFile},${outputFile}`, () => {
19       const i = require(inputFile);
20       const o = require(outputFile);
21
22       const actual = alg(i);
23
24       try {
25         assert.deepEqual(actual, o);
26         return save(outputFile, actual);
27       } catch (ex) {
28         return save(outputFile, actual).then(
29           () => Promise.reject(ex)
30         );
31       }
32     });
33   }
34 });
```

11-testing/samples/snapshot/test/spec.js

Snapshot testing in mocha

sample implementation

- mam snapshot na gite
- pustim test
- snapshot sedi
- zmenim imlementaciu
- snapshot nesedi
- je to očakávaná zmena ?
- commitnem snapshot a pustím test
- nie je to očakávaná zmena ? snapshot chceckoutnem, nájdem problém a spravím fix a pustím test

```
$ npx mocha --colors snapshot/test/spec.js

  Snapshot test
    √ unroll
    √ snapshot: ./data/01.in.json,./data/01.out.json

  2 passing (17ms)

$ npx mocha --colors snapshot/test/spec.js

  Snapshot test
    √ unroll
    1) snapshot: ./data/01.in.json,./data/01.out.json

$ git diff
diff --git a/2018-javascript/pre
esting/samples/snapshot/test/dat
index a6c1ba0..b4c5dcf 100644
--- a/2018-javascript/prednasky/
+++ b/2018-javascript/prednasky/
@@ -1,4 +1,5 @@
 {
   "a": 10,
-  "b": 20
+  "b": 20,
+  "c": 30
 }

$ git commit -am "[CHR] alg now supports new sum property"
[master c6f59a7] [CHR] alg now supports new sum property
 2 files changed, 3 insertions(+), 2 deletions(-)

$ npx mocha --colors snapshot/test/spec.js

  Snapshot test
    √ unroll
    √ snapshot: ./data/01.in.json,./data/01.out.json

  2 passing (17ms)
```

Self Testing Code

Do I really need the frameworks

Self Testing Code

Lax definitions:

- Self-Testing Code is the name I used in Refactoring to refer to the practice of writing comprehensive automated tests in conjunction with the functional software.
- When done well this allows you to invoke a single command that executes the tests - and you are confident that these tests will illuminate any bugs hiding in your code
- https://martinfowler.com/bliki/SelfTestingCode.html

Pre nás:

Kód ktorý obsahuje testy priamov z zdrojákoch daného modulu,

- testy sa vykonajú pri loade modulu,
- ak popadajú modul sa nenaloaduje
- a systém sa nespustí

niečo na spôsob POST:

- A power-on self-test (POST) is a process performed by firmware or software routines immediately after a computer or other digital electronic device is powered on.

example

- test code in the same file as module
  - quicker access to test to get understanding of code or to modify test
- test function loaded and evaluated only when needed
  - sensitivity to ENV variable, custom or NODE_ENV
  - IIFE
  - minimal test framework dependency (assert, console.assert)
- package json
  - just skelet of idea

```javascript
// generic composition
const pipe = (...fns) =>
  x => fns.reduce((v, f) => f(v), x);

// function pipe(...fns) {
//   // TODO: rewrite above reduce to for cycle
// }
module.exports = pipe;





// ------------- TESTS ------------------------------
process.env.SELF_TEST && (() => {
  console.error(`[self test]:${__filename}:...`)

  const assert = require("assert");

  const a = (v) => `a(${v})`
  const b = (v) => `b(${v})`
  const c = (v) => `c(${v})`

  assert.equal(pipe(a, b, c)("x"), "c(b(a(x)))");

  console.error(`[self test]:${__filename}:OK`)
})();
```

```
"scripts": {
  "test": "SELF_TEST=1 node -r ./src/pipe.js -r ./src/max.js |-p '\"done\"' "
},
```

Code Coverage

Code Coverage

- In computer science, test coverage is a measure used to describe the degree to which the source code of a program is executed when a particular test suite runs. A program with high test coverage, measured as a percentage, has had more of its source code executed during testing, which suggests it has a lower chance of containing undetected software bugs compared to a program with low test coverage.

https://en.wikipedia.org/wiki/Code_coverage

Code Coverage

- Odpovedá na otázku „Koľko ***percent kódu*** je pokrytého ***daným testom***“
  - riadkov
  - funkcií
  - statementov
  - *vetiev* kódu

```
1  module.exports = {
2    a: function(x) {
3      if (x) {
4        return x;
5      } else {
6        return 0;
7      }
8    }
9  }
```

```
4  describe("", () => {
5    it("a1", function() {
6      assert.equal(m.a(null), 0);
7    });
8    it("a2", function() {
9      assert.equal(m.a(10), 10);
10   });
11 });
```

```
  2 passing (4ms)

----------|----------|----------|----------|----------|-------------------|
File      |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
----------|----------|----------|----------|----------|-------------------|
All files |      100 |      100 |      100 |      100 |                   |
 m.js     |      100 |      100 |      100 |      100 |                   |
----------|----------|----------|----------|----------|-------------------|
```

Code Coverage

- Odpovedá na otázku „Koľko ***percent kódu*** je pokrytého ***daným testom***“
  - riadkov
  - funkcií
  - statementov
  - *vetiev* kódu

```
1  module.exports = {
2    a: function(x) {
3      if (x) {
4        return x;
5      } else {
6        return 0;
7      }
8    }
9  }
```

```
4  describe("", () => {
5    it("a1", function() {
6      assert.equal(m.a(null), 0);
7    });
8    // it("a2", function() {
9    //   assert.equal(m.a(10), 10);
10   // });
11 });
```

```
  1 passing (4ms)

----------|----------|----------|----------|----------|-------------------|
File      |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
----------|----------|----------|----------|----------|-------------------|
All files |       75 |       50 |      100 |       75 |                   |
 m.js     |       75 |       50 |      100 |       75 |                 4 |
----------|----------|----------|----------|----------|-------------------|
```

Tools for Code Coverage

- https://istanbul.js.org
  - Istanbul instruments your ES5 and ES2015+ JavaScript code with line counters, so that you can track how well your unit-tests exercise your codebase.
  - The nyc command-line-client for Istanbul works well with most JavaScript testing frameworks: tap,mocha, A VA, etc.
- Demo: 11-cvicenie/

All files **index.js**

96.3% Statements 26/27 **90%** Branches 9/10 **100%** Functions 8/8 **95.83%** Lines 23/24

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

```
1      //
2   1x const { Transform } = require("stream");
3
4   1x module.exports = {
5        add: function() {
6   4x     return new AddBom();
7        }
8      }
9
10  1x const bom = Buffer.from([0xEF, 0xBB, 0xBF]);
11  1x const bufLength = (bufs) =>
12 17x   bufs.reduce((a, b) => a.length || 0 + b.length, 0);
13  4x const hasBom = (buf) => buf.slice(0, 3).equals(bom);
14
15     class AddBom extends Transform {
16
17       constructor() {
18  4x     super();
19  4x     this._bomDone = false;
20  4x     this._buff = [];
21       }
22       _transform(chunk, enc, cb) {
23  7x     I if (this._bomDone)
24           return cb(null, chunk);
25
26  7x     this._buff.push(chunk);
27  7x     if (bufLength(this._buff) >= 3)
28  2x       this._pushBuffered();
29
30  7x     cb();
31       }
32       _flush(cb) {
33  4x     if (!this._bomDone)
34  2x       this._pushBuffered();
35  4x     cb();
36       }
37       _pushBuffered() {
38  4x     let chunk = Buffer.concat([...this._buff]);
39  4x     if (!hasBom(chunk)) this.push(bom);
40  4x     this.push(chunk);
41  4x     this._bomDone = true;
42  4x     this._buff = null;
43       }
44     }
```

Code Coverage a špeciatilty JS

- Niektoré konštrukcie v JS píšeme inak
- Výsledky nemusia zodpovedať *očakávaniam*

```
1  module.exports = {
2    a: function(x) {
3      if (x) return x;
4      else return 0;
5    },
6    b: function(x) {
7      return x || 0;
8    }
9  }
```

```
4  describe("", () => {
5    it("", () => {
6      assert.equal(m.a(null), 0);
7      assert.equal(m.a(10), 10);
8
9      assert.equal(m.b(null), 0);
10     //assert.equal(m.b(10), 10);
11   });
```

- Asi by som nečakal 100% branch coverage

```
 1 passing (4ms)

----------|----------|----------|----------|----------|-------------------|
File      |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
----------|----------|----------|----------|----------|-------------------|
All files |      100 |      100 |      100 |      100 |                   |
 m.js     |      100 |      100 |      100 |      100 |                   |
----------|----------|----------|----------|----------|-------------------|
```

Code Coverage Instrumentation

- Aby sa dala robiť coverage, tooling spravidla spraví instrumentation kódu
- Spraví si mapy funkcií statementov a branches
- Na stanovené miesta si podopĺňa countre

```
var cov_2d9ixqu4rm = function() {
    //...
    coverageData = {

      statementMap: {
        "0": { start: { line: 1, column: 0 }, end: { line: 5, column: 1 } },
        "1": { start: { line: 3, column: 4 }, end: { line: 3, column: 18 } }
      },
      fnMap: {
        "0": { name: "(anonymous_0)", decl: { start: { line: 2, column: 5 },
      },
      branchMap: {
        "0": {
          loc: { start: { line: 3, column: 11 }, end: { line: 3, column: 17
          type: "binary-expr",
          locations: [{ start: { line: 3, column: 11 }, end: { line: 3, colu
          line: 3
        }
      },
      s: { "0": 0, "1": 0 },
      f: { "0": 0 },
      b: { "0": [0, 0] },
      _coverageSchema: "43e27e138ebf9cfc5966b082cf9a028302ed4184"
    },
    //....
}();
cov_2d9ixqu4rm.s[0]++;
module.exports = {
  b: function(x) {
    cov_2d9ixqu4rm.f[0]++;
    cov_2d9ixqu4rm.s[1]++;
    return (cov_2d9ixqu4rm.b[0][0]++, x) || (cov_2d9ixqu4rm.b[0][1]++, 0);
  }
};
```

Príklad z nyc nástroja https://github.com/istanbuljs/nyc

Performance Testing

Benchmarking

- Potrebujeme odmerať a porovnať, koľko nám trvá daná operácia nad:
- Rôznymi dátami
- Tými istými dátami kódnutá inak
- V Browsri A vs. Browsri B
- ....

Benchmarks

- Writing JavaScript benchmarks isn't as simple as it seems. Even without touching the subject of potential cross-browser issues, there are a lot of pitfalls — booby traps, even — to look out for.
  - https://calendar.perfplanet.com/2010/bulletproof-javascript-benchmarks/

- https://benchmarkjs.com A benchmarking library that supports high-resolution timers & returns statistically significant results
- …

Tools for benchmarking js code

- https://github.com/bestiejs/benchmark.js
- https://jsperf.com/

- https://perf.zone
- http://jsbench.github.io
- …
- …
- http://jsben.ch

How ***we*** have conributed:

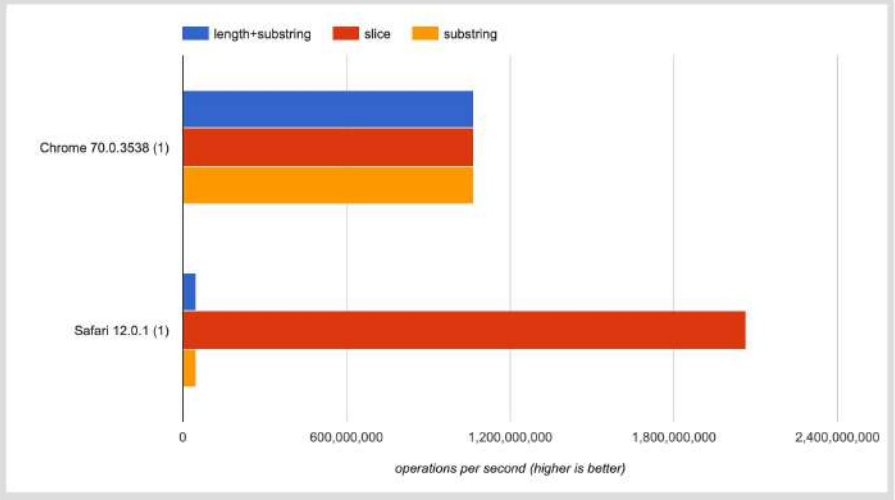
- https://jsperf-playground.herokuapp.com
- https://jsperf.com/browse/ainthek

Príklady

- Jedna vec sa dá v JS kódnuť viacerými spôsobmi
- Napríklad posledných 5 znakov z reťazca
- A ozaj nie je jedno, či napíšete ***slice*** alebo ***substring***, teda aspoň v Safari to nie je jedno

Testing in Safari 12.0.1 / Mac OS X 10.14.1

Test		Ops/sec
slice	`s.slice(-5);`	ready
substring	`s.substring(s.length-5,s.length);`	ready
length+substring	`let l=s.length` `s.substring(l-5,l);`	ready



https://jsperf.com/string-tail/2

Writing and interpreting micro benchmarks

Using jsPerf correctly: (opatrne 8 rokov stare)

- https://www.youtube.com/watch?v=RLbAKxCAdI8&t=434s
- https://www.slideshare.net/mathiasbynens/using-jsperf-correctly/

Vyacheslav Egorov - Performance and benchmarking (opatrne silne „populisticke“):

- https://www.youtube.com/watch?v=65-RbBwZQdU
- https://www.youtube.com/watch?v=g0ek4vV7nEA

Benchmark.js in node.js

- https://benchmarkjs.com/docs
  - Benchmark
  - Suite
  - Events
- Compare ***the comparables***
  - Absolute
  - ***Relative***
  - Extract to setup and teardown
- See previous and next slide
- Ako by sa zmenil pomer keby v array bolo 100000 itemov ?

```
suite.add('sort(a-b).reverse()', function() {
  let x = a.sort((a, b) => a - b).reverse();
  //assert.deepEqual(x, r);
})
suite.add('sort(b-a)', function() {
  let x = a.sort((a, b) => b - a);
  //assert.deepEqual(x, r);
})
suite.add('sort(i * a - i * b).reverse()', function() {
  let i = -1;
  let x = a.sort((a, b) => i * a - i * b);
  //assert.deepEqual(x, r);
})
```

No asserts

```
__noop x 1,037,989,429 ops/sec ±0.93% (92 runs sampled)
sort(a-b).reverse() x 899,267 ops/sec ±0.35% (91 runs sampled)
sort(b-a) x 3,044,844 ops/sec ±0.33% (96 runs sampled)
sort(i * a - i * b).reverse() x 2,811,646 ops/sec ±0.41% (96 runs sampled)
```

Expect **reasonable** code to have **reasonable** perforance

???

Expect
reasonable code
to have
reasonable perforance

is version 1. or 3. really reasonable code ?

```
suite.add('sort(a-b).reverse()', function() {
  let x = a.sort((a, b) => a - b).reverse();
  assert.deepEqual(x, r);
})
suite.add('sort(b-a)', function() {
  let x = a.sort((a, b) => b - a);
  assert.deepEqual(x, r);
})
suite.add('sort(-1*a-1*b).reverse()', function() {
  let i = -1;
  let x = a.sort((a, b) => i * a - i * b);
  assert.deepEqual(x, r);
})
```

```
$ node test/sort-reverse.js
__noop                    x        1,034,749,698
sort(a-b).reverse()       x        898,063
sort(b-a)                 x        3,184,151 ±0.38%
sort(-1*a-1*b).reverse()  x        2,881,135
```

Mutation Testing

Mutation Testing

- Mutation testing (or mutation analysis or program mutation) is used to
  - design new software tests and
  - evaluate the quality of existing software tests.
- Mutation testing involves modifying a program in small ways. Each mutated version is called a mutant and tests detect and reject mutants by causing the behavior of the original version to differ from the mutant. This is called killing the mutant.
  - Test suites are measured by the percentage of mutants that they kill.
  - New tests can be designed to kill additional mutants.
- Mutants are based on well-defined mutation operators that
  - either mimic typical programming errors (such as using the wrong operator or variable name) or
  - force the creation of valuable tests (such as dividing each expression by zero).
- The purpose is to help the tester develop effective tests or locate weaknesses in the test data used for the program or in sections of the code that are seldom or never accessed during execution. Mutation testing is a form of white-box testing.
- manual
- automated

Mutation testing - before refactor

- **Story:** download lib, review, bad code found, I want to refactor, be sure tests will be ok and then make pull request
- **Process:** mutate code by hands, if tests red, lucky, if tests green, author is lame (double lame, bad code, bad tests)
- Real life example: test is missing coverage for lines in point 4)

1. clone/install

```
git clone https://github.com/gomfunkel/node-exif.git
cd node-exif
npm install
npm test
```

2. test

```
node-exif API
  √ test constructor (filename)
  √ test constructor (buffer)
  √ test loadImage (filename)
  √ test loadImage (buffer)
  √ test wrapper

node-exif tests
  √ test agfa-makernotes.jpg
  √ test down-mirrored.jpg
  √ test evil1.jpg
  √ test lens_info.jpg
  √ test right.jpg
  √ test short-ascii-II.jpg
  √ test short-ascii-MM.jpg
  √ test sony-alpha-6000.jpg
  √ test test1.jpg
  √ test test2.jpg
  √ test test3.jpg


16 passing (75ms)
```

3. Codereview

```
151     fs.readFile(image, function (error, data) {
152       if (error) {
153         callback(new Error('Encountered the following erro
154         return;
155       }
156       self.processImage("File: "+image, data, callback);
157     });
158     return;
159   }
160
161   callback(new Error('Given image is neither a buffer nor
162 };
163
164 ExifImage.prototype.processImage = function (source, data,
165   assert(typeof(source)===="string", "Source must be a stri
166   assert(typeof(callback)===="function", "Callback must be
167
168   var offset = 0;
169
170   if (data[offset++] != 0xFF || data[offset++] != 0xD8) {
171     var e=new Error('The given image is not a JPEG and thu
```

4. “mutate” (rem all these lines)

```
170   if (data[offset++] != 0xFF || data[offset++] != 0xD8) {
171     // var e=new Error('The given image is not a JPEG and t
172     // e.source=source;
173     // e.code="NOT_A_JPEG";
174     // callback(e);
175     // return;
176   }
```

5. test (shell fail!)

```
node-exif API
  √ test constructor (filename)
  √ test constructor (buffer)
  √ test loadImage (filename)
  √ test loadImage (buffer)
  √ test wrapper

node-exif tests
  √ test agfa-makernotes.jpg
  √ test down-mirrored.jpg
  √ test evil1.jpg
  √ test lens_info.jpg
  √ test right.jpg
  √ test short-ascii-II.jpg
  √ test short-ascii-MM.jpg
  √ test sony-alpha-6000.jpg
  √ test test1.jpg
  √ test test2.jpg
  √ test test3.jpg


16 passing (75ms)
```

6. DOOM

Mutation testing - before refactor

- Mutation testing revealed that tests are ***missing coverage for important code part***
- Real life example: to fix the code I need to add tests first

1. Add missing test code

```
43  it('test loadImage (filename)', function(done) {
44
45    var ExifImage = require('..').ExifImage;
46
47    var exif=new ExifImage();
48
49    exif.loadImage(path, function(error, data) {
50      if (error) {
51        throw error;
52      }
53
54      assert.equal(JSON.stringify(data), json, "Not same datas ?");
55
56      done();
57    });
58  });
59
60  it('test loadImage (filename), not a JPEG', function(done) {
61
62    var ExifImage = require('..').ExifImage;
63    var exif=new ExifImage();
64    // reading self (JS file which shell fail)
65    exif.loadImage(__filename, function(error, data) {
66      if (error) {
67        assert(error.code=="NOT_A_JPEG");
68        done();
69      }
70      else {
71        done(new Error("Unexpected succes"));
72      }
73    });
74  });
```

2. test

```
node-exif API
  √ test constructor (filename)
  √ test constructor (buffer)
  √ test loadImage (filename)
  √ test loadImage (filename), not a JPEG
  √ test loadImage (buffer)
  √ test wrapper

node-exif tests
  √ test agfa-makernotes.jpg
  √ test down-mirrored.jpg
  √ test evil1.jpg
  √ test lens_info.jpg
  √ test right.jpg
  √ test short-ascii-II.jpg
  √ test short-ascii-MM.jpg
  √ test sony-alpha-6000.jpg
  √ test test1.jpg
  √ test test2.jpg
  √ test test3.jpg


17 passing (84ms)
```

4. test

```
node-exif API
  √ test constructor (filename)
  √ test constructor (buffer)
  √ test loadImage (filename)
  1) test loadImage (filename), not a JPEG
  √ test loadImage (buffer)
  √ test wrapper

node-exif tests
  √ test agfa-makernotes.jpg
  √ test down-mirrored.jpg
  √ test evil1.jpg
  √ test lens_info.jpg
  √ test right.jpg
  √ test short-ascii-II.jpg
  √ test short-ascii-MM.jpg
  √ test sony-alpha-6000.jpg
  √ test test1.jpg
  √ test test2.jpg
  √ test test3.jpg


16 passing (95ms)
1 failing
```

3. “mutate” (rem all these lines)

```
170  if (data[offset++] != 0xFF || data[offset++] != 0xD8) {
171    // var e=new Error('The given image is not a JPEG and t
172    // e.source=source;
173    // e.code="NOT_A_JPEG";
174    // callback(e);
175    // return;
176  }
```

5. COOL
Test detected
mutation

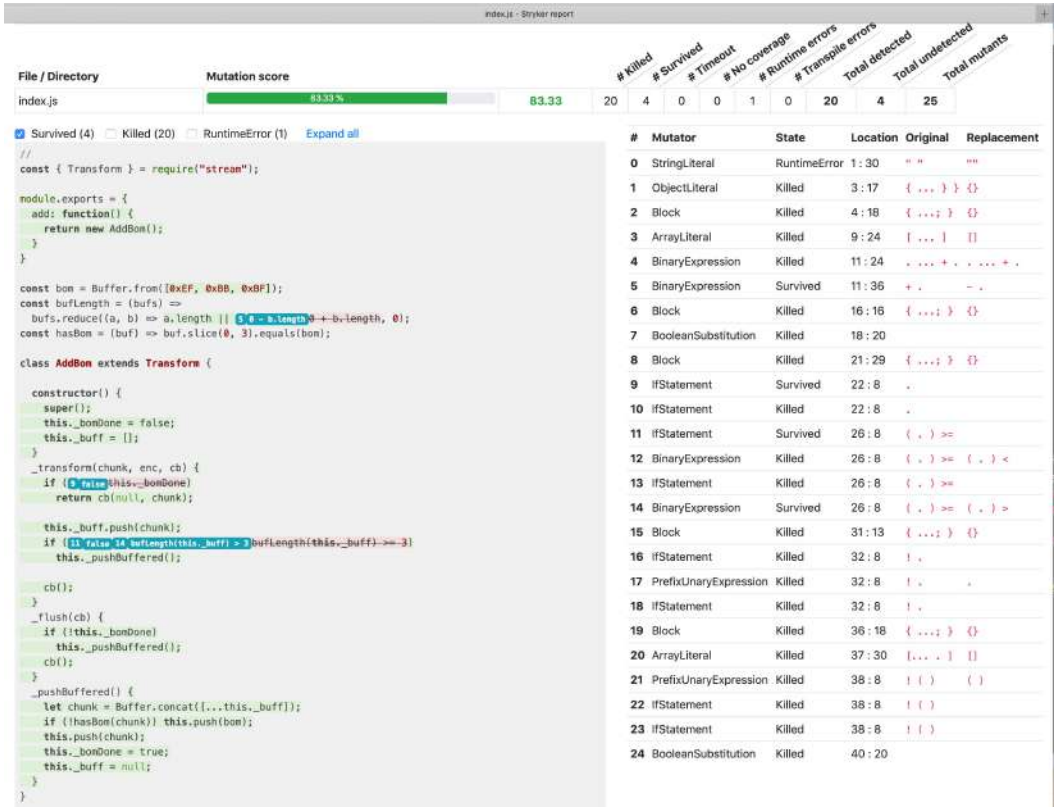
Mutation testing - before refactor

- **Tests added for missing code branches**
  - If needed repeat previous, add more tests
- Refactor
  - Optimize loadImage and processImages codes
- Pull request – tests
- Pull request - optimization

```
    fs.readFile(image, function (error, data) {
      if (error) {
        callback(new Error('Encountered the following erro
        return;
      }
      self.processImage("File: "+image, data, callback);
    });
    return;
  }

  callback(new Error('Given image is neither a buffer nor
};

ExifImage.prototype.processImage = function (source, data,
  assert(typeof(source)=="string", "Source must be a stri
  assert(typeof(callback)=="function", "Callback must be

  var offset = 0;

  if (data[offset++] != 0xFF || data[offset++] != 0xD8) {
    var e=new Error('The given image is not a JPEG and thu
```

Tools for Mutation Testing

- https://stryker-mutator.io/
- Demo: 11-cvicenie/



Mutation Testing a špeciatilty JS

- JS je špecifický, potrebuje špecifické mutátory,
- základná sada (napríklad v stryker tool-e) pokrýva len *spoločné* základné mutátory
- Ďalšie zaujímavé a JS špecifické by boli
  - Regex mutator
  - Timers mutators
  - Sync/async callback mutators
  - ...
  - ...

Mutator	Stryker	Stryker.NET	Stryker4s
Binary Operators	✅	✅	ℹ️[1]
Boolean Substitutions	✅	✅	ℹ️[2]
Logical operators	✅	✅	✅
Unary operators	✅	✅	❌
Update operators	✅	✅	n/a
Remove conditionals	✅	❌	❌
Assignment mutator	❌	✅	n/a
Array declarator	✅	❌	❌
String mutator	✅	❌	✅
Block statement	✅	❌	❌
Checked mutator	n/a	✅	n/a
Method mutator	n/a	❌	✅

Test packages and package.json

- Test framework libraries and assert libraries shell be specified as
  - **devDependencies**
  - not dependencies
- Unless you extend the framework itself

- Example from npm: mocha usage on npm

```
$ wc -l out/mocha-as-*
   4357 out/mocha-as-dependency.txt
 139887 out/mocha-as-devDependency.txt
 144244 total
```

data from my *npma project*

TODO: mocking libraries

_aux

Assert API

je lepšie písať:

assert(x===1) alebo assert.equal(x,1) ???

- Vždy as snažte aby bol report dobrý, na syntaxi až tak nezáleží
- Akékoľvek ***fancy a fluent asserty*** sú mi nanič ak reportujú zle (nečitateľne), alebo nemajú presne definované správania

Cieľ:

- test čitateľný
- aj report čitateľný
- jedno na úkor druhého nemá zmysel

Assert API

- prefer **assert.equal**(r,e) over assert(r==e)
- You will get **nicer errors**
- Simple refactoring, but only for clear situations where literal is used, otherwise you do not know what is expected and what is real value
- Note: ak používate better-assert je to jedno, ten vyextrahuje presné znenie kódu assertu

```
it("assert(x == 20)", () => {
    let x = 10;
    assert(x == 20);
});
it("assert.equal(x, 20)", () => {
    let x = 10;
    assert.equal(x, 20);
});
```

```
1) test display of assertion errors
     assert(x == 20):

    AssertionError [ERR_ASSERTION]: The expression evaluated to a falsy value:

assert(x == 20)

    + expected - actual

    -false
    +true


2) test display of assertion errors
     assert.equal(x, 20):

    AssertionError [ERR_ASSERTION]: 10 == 20
    + expected - actual

    -10
    +20
```

```
$ echo "assert(x==3)" | grasp -e 'assert($x==_literal)' -R 'assert.equal({{$x}}, {{_literal}})'
assert.equal(assert, 3)
```

_aux

- https://medium.com/@the1mills/how-to-test-your-npm-module-without-publishing-it-every-5-minutes-1c4cb4b369be

- https://www.youtube.com/watch?v=65-RbBwZQdU

Configuration testing

Configuration testing

- TODO: definition

example - node.js stream

- Experiment with buffer sizes
  - measure memory
  - Throughput
- Expose your application stream parameters as configuration
  - To fine-tune on target platform, data etc.
- Implement adaptive code

example - node.js stream

- Buffers and poolSize
- Buffers and --zero-fill-buffers