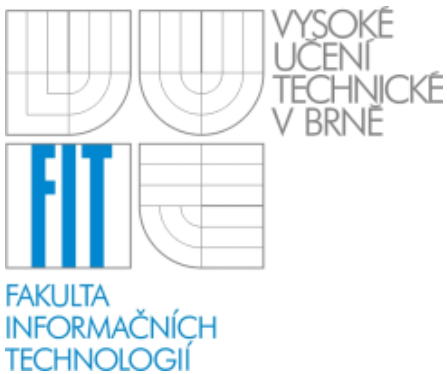
Testování Software

Petr Škoda

Ústav počítačové grafiky a multimédií
Fakulta informačních technologií

Vysoké učení technické v Brně



Testování

- Součást procesu vývoje
- Proces užívaný k odhalení defektů v softwaru a prokázání, že software dosáhl specifikované úrovně kvality vzhledem k vybraným atributům
- **Verifikace** – vyhodnocování, zda software v dané vývojové fázi splňuje specifikace (odpovídá tomu, jak jsme jej naplánovali)
  „stavíme tu věc správně?“
- **Validace** – vyhodnocování, zda software splňuje požadavky (dělá to co od něho bylo očekáváno)
  „postavili jsme správnou věc?“

Způsob testování

Manuální

- scénář, vstupní a výstupní data
- člověk realizuje daný scénář

Automatické

- test je realizován programem
- jsou znovuspustitelné

Černá skříňka (black box)

- bez znalosti vnitřní struktury
- důležité jsou pouze vstupy a výstupy
- ohled pouze na specifikaci/funkci

Průhledná skříňka (white/glass box)

- s ohledem na vnitřní strukturu
- prakticky testování napsaného kódu
- snaha pokrýt všechny větve
- může být časově náročné; příliš detailní

Úrovně testů

- **Jednotkové (Unit)** – jednotlivá komponenta
- **Integrační** – několik komponent jako skupina
- **Systémové** – celý systém s ohledem např. na spolehlivost či výkonnost

Jednotkové testy

- *Jednotka* – záleží na paradigmatu užívaného jazyka (funkce/objekt/...)
- Může to být i komponenta třetí strany
- Testy jednotky musí probíhat v izolaci od ostatních
- Cílem je zjistit, zda jednotka funguje dle specifikace (verifikace)

xUnit

- Framework pro unit testy pro různé jazyky (platformy) postavené na designu SUnit pro *Smalltalk* [2]
- Poskytují konstrukce pro vytváření opakovaně spustitelných testů a ověřovaní očekávaného chování
- Umožňují shlukovat související testy
- Java, Scala – jUnit
- .NET – nUnit
- C++ – cppUnit
- Python – pyUnit
- http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

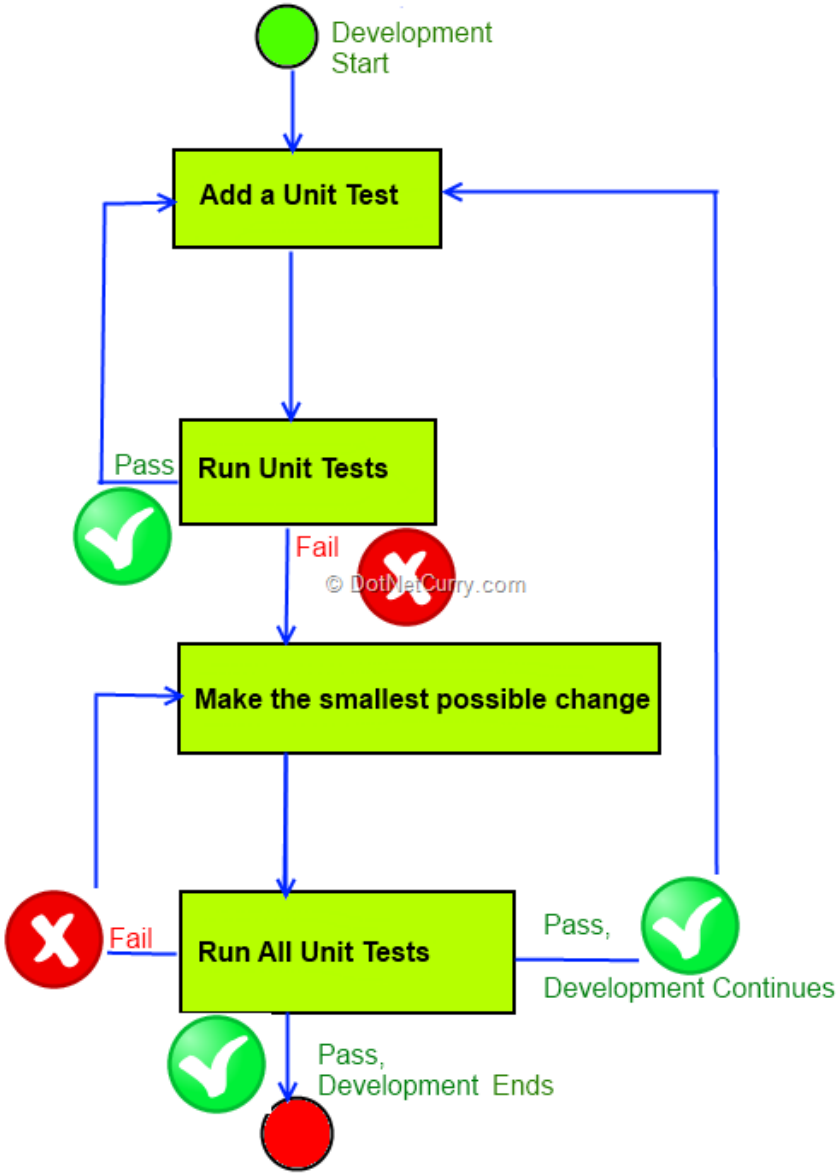
TestCase

- Shlukuje sadu testů (metod) testujících jednu jednotku
- metoda *setup* – vytvoření inventáře a zajištění předpokladů (preconditions)
- spuštění samotného testu, který obsahuje jedno či více tvrzení (assertions), které mají platit
- metoda *teardown* – uvedení do původního stavu tak, aby nebyly ovlivněny další testy
- *setup/teardown* se spouští před/po každém testu (zajišťuje izolaci)

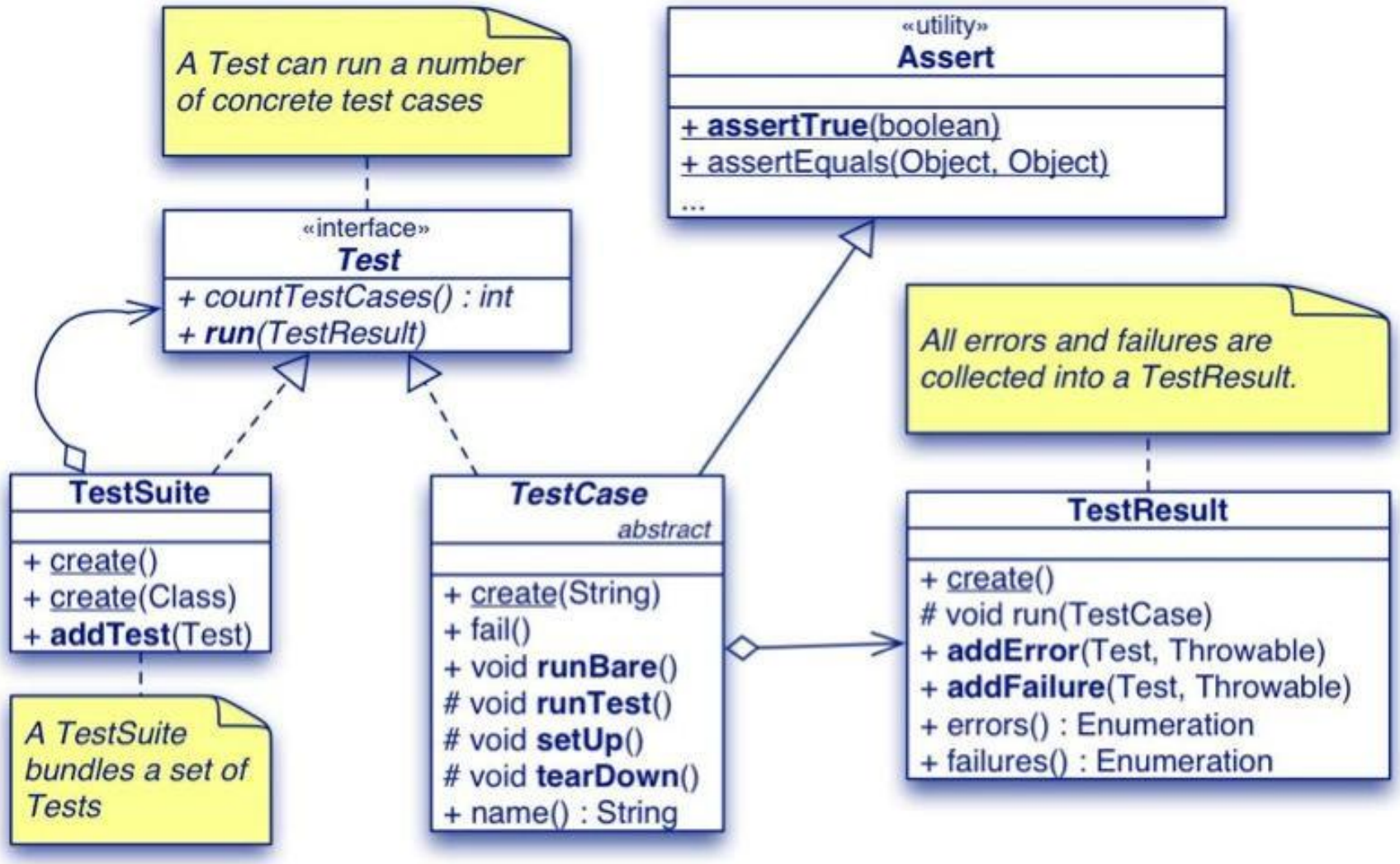
TestSuite

- Sada testovacích případů (test case)
- Např. test všech jednotek v konkrétním subsystému

Koncepce xUnit



jUnit v UML


A Test can run a number of concrete test cases
«utility»
Assert
+ assertTrue(boolean)
+ assertEquals(Object, Object)
...
«interface»
Test
+ countTestCases() : int
+ run(TestResult)
All errors and failures are collected into a TestResult.
TestSuite
+ create()
+ create(Class)
+ addTest(Test)
TestCase
abstract
+ create(String)
+ fail()
+ void runBare()
# void runTest()
# void setUp()
# void tearDown()
+ name() : String
TestResult
+ create()
# void run(TestCase)
+ addError(Test, Throwable)
+ addFailure(Test, Throwable)
+ errors() : Enumeration
+ failures() : Enumeration
A TestSuite bundles a set of Tests

Fixture

- *Inventář*
- Množina dat a objektů
- Tvoří konfiguraci testu
- Může obsahovat
  - mock objekty
  - vstupní a očekávané hodnoty
  - inicializace databáze vhodnými hodnotami
  - vstupní soubory, apod.

Příklad sady testů (Ruby)

```ruby
class BankAccountTest < Test::Unit::TestCase def
   setup
      @account = BankAccount.new
   end

   def test_initBalance
      assert_equal( @account.balance, 0 ) end

   def test_store
      @account.store( 100 )
      assert_equal( @account.balance, 100 ) end

   def test_withdraw
      @account.withdraw( 100 )
      assert_equal( @account.balance, -100 ) end
end
```

Izolace

- Reálné funkce využívají jiné funkce
- Analogicky to platí pro objekty v objektovém paradigmatu
- Aby se zabránilo průniku chyb odjinud nesmí se používat kód jiné jednotky
- Používají se simulace okolí
- *Stub (pahýl)* – simuluje jiný kód pro účely testu
  - může vracet předdefinované hodnoty
  - může zaznamenávat své volání
- *Mock (napodobenina)*
  - ověřuje chování toho, kdo ho používá
  - jsou ověřovány interakční scénáře

Příklad mock objektu v jMock 2

```
// objekt, ktery vyrabi napodobeniny
Mockery context = new Mockery();

// mock objekt - napodobenina
final AccountManager accountManager = context.mock(AccountManager.class);

final Account accountA = new Account(1);
final Account accountB = new Account(2);

// nastavime ocekavane chovani
context.checking(new Expectations() {{
        allowing (accountManager).getAccountById(1);
        will (returnValue(accountA));
        allowing (accountManager).getAccountById(2);
        will (returnValue(accountB));

        // ocekava se volani saveAccount jednou na kazdem uctu
        oneOf(accountManager).saveAccount(accountA);
        oneOf(accountManager).saveAccount(accountB);
}});

accountManager.getAccountById("1") // => accountA
accountManager.getAccountById("2") // => accountB
```

Integrační testy

- Jednotky tvoří podsystémy
- Integrační testy testují tyto podsystémy
- Jinými slovy napodobeniny jsou nahrazeny skutečnými implementacemi
- Cílem je detekovat chyby na rozhraní jednotek a jejich interakce

Systémové testy

- Testování systému jako celku
- Ujištění se, že funguje dle požadavků
- Typy systémových testů
  - funkční
  - výkonnostní
  - zátěžové
  - bezpečnostní
  - konfigurační
  - zaměřené na zotavení
- Akceptační testy
  - prakticky systémové testy při předávání zákazníkovi
  - ověřuje splnění požadavků zákazníka
  - black box

Refaktoring

- Změna v systému, která nemění chování, ale vylepšují jiné kvality jako jednoduchost, flexibilitu nebo srozumitelnost
- Je těžké navrhnout a naprogramovat netriviální systémy na první pokus
- Změny jsou nevyhnutelné

Regresní testování

- Testování prováděné po změnách kódu
- Potvrzení, že nový kód odpovídá specifikaci
- Potvrzení, že jiná funkcionalita nebyla změnou ovlivněna
- *Black box* testy zde vlastně hrají roli specifikace
- I proto jsou dobré automatizované testy

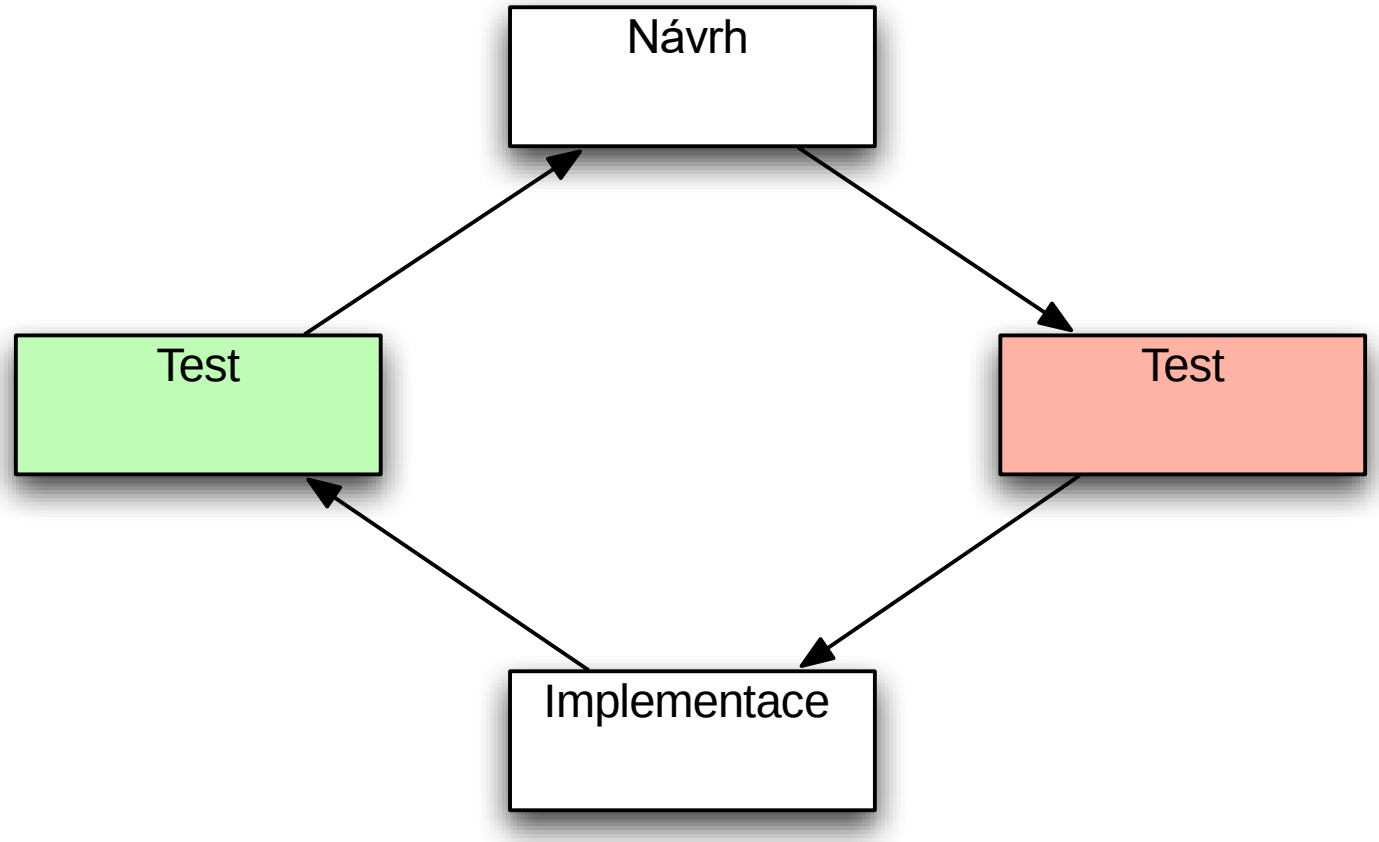
Vývojář vs. Tester

- Vývojář kód píše, tedy ho důvěrně zná ze svého pohledu
- Kód je psán na základě určitých předpokladů, ty může vývojář brát jako jisté, přičemž nejsou
- Jím psané testy proto nemusí být dostatečně vyčerpávající (můžou v nich být stejné předpoklady)
- Ideální tester se nepodílel na vývoji – přináší novou perspektivu
- Na integrační a systémové úrovni mohou působit celé týmy testerů

Test-driven development

- Proces vývoje software
- Řadí se k agilním procesům
- Podporuje rychlé iterace, adaptabilitu na požadavky a zaměření se na to, co je skutečně potřeba
- Testy hrají centrální roli – slouží jako specifikace

Proces vývoje konceptúalně



Iterace

1. Napsat test
   - prakticky návrh rozhraní (API)
2. Ujistit se, že test neprojde
3. Napsat kód tak, aby test prošel
   - minimální kód pro splnění testu
4. Ujistit se, že projde celá sada testů
   - potom způsob jakým byl kód napsán nenarušil kód ověřovaný jinými testy
5. Refaktorizace (pročištění) kódu s průběžným ověřováním testů

Výhody

- Všechen produkční kód je testovaný
- Je implementována pouze potřebná funkcionalita
- Průběžným spouštěním testů je možno zamezit dlouhým úpravám kódu bez znalosti stavu věcí
- Občas může být výhodnější vrátit se do stavu, kdy testy prošly než hledat a odstraňovat chyby

Behaviour Driven Development

- Rozšíření TDD
- Slovní popis testů, tudíž snadněji srozumitelný
- Testy skutečně popisují očekávané chování, tedy specifikaci kódu
- Testy odpovídají „*příběhům uživatelů*“ („*users' stories*“), které jsou často výchozím bodem agilních metod vývoje software
- *RSpec* (Ruby), *NSpec* (.Net), *JBehave* (Java)

BDD v RSpec

```
describe "bank account" do
  before(:each) do
    @account = BankAccount.new
  end

  it "should have balance 0 after creation" do
    @account.balance.should == 0
  end

  it "storing amount of money should increase balance" do
    @account.store(100)
    @account.balance.should == 100
  end
end
```

Mock objekt v RSpec

```
require "account"

describe Account do context
   "when closed" do
      it "logs an account closed message" do
         logger = double("logger")
         account = Account.new
         account.logger = logger

         logger.should_receive(:account_closed)

         account.close
      end
   end
end
```

Continuous Integration

- Průběžná integrace všech částí systému
- Jediné úložiště zdrojů potřebných pro sestavení, tj. systém pro správu verzí (Mercurial, Git, CVS etc.)
- Automatické sestavení po každém potvrzení (*commit*)
- Testy součástí každého sestavení
- Urychluje odhalení a opravy chyb a konfliktů
- Testování v klonu produkčního prostředí
- *Buildbot, Hudson, CruiseControl*

Časté problémy testů

- Nesprávné předpoklady
- Nekompletní testovací případ
- Nesprávně testována integrace
- Nedostatečné testování výkonu

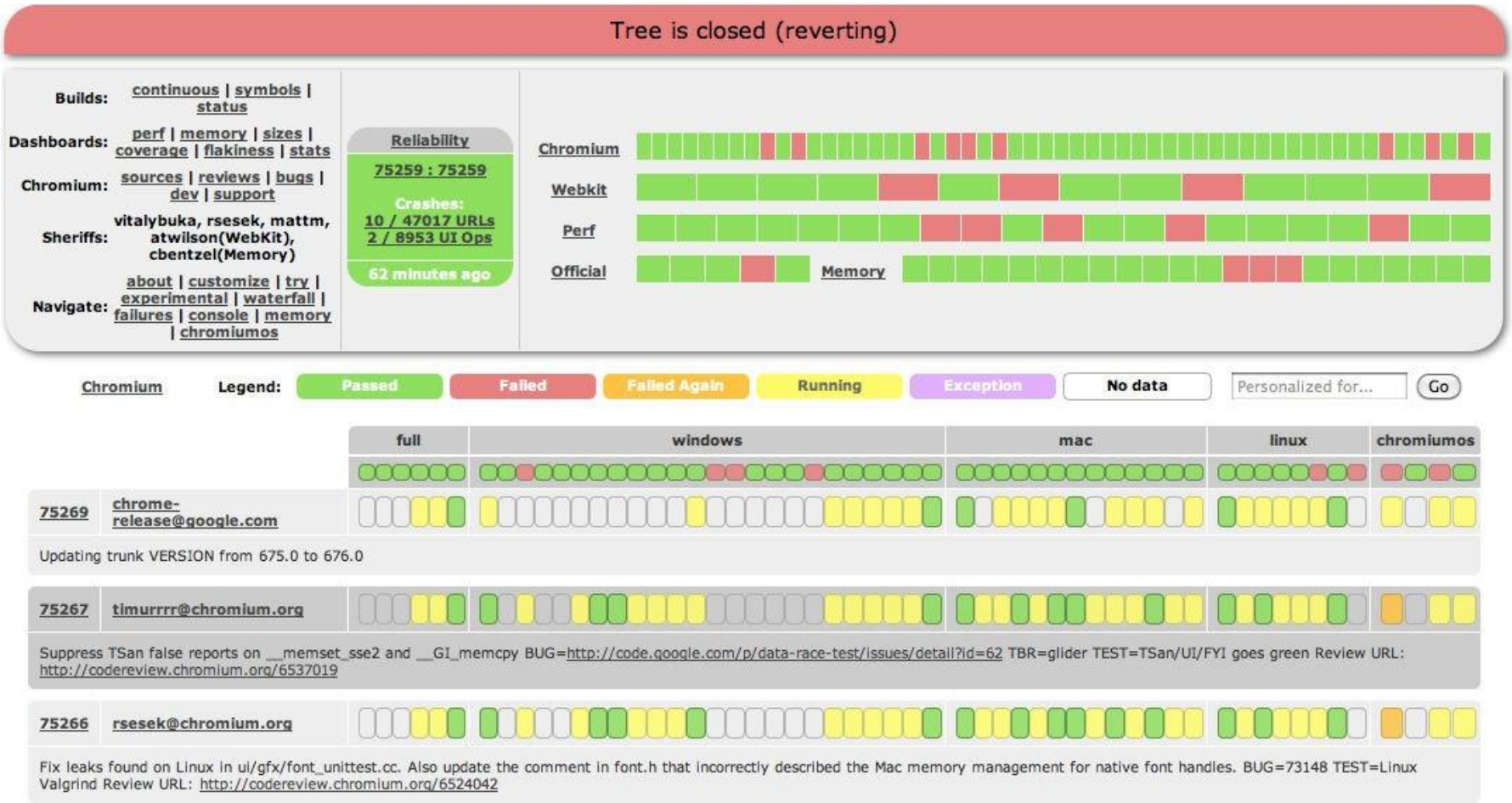
Časté výmluvy vývojářů

- It's working fine on staging
- We didn't have enough time to test
- It's okay, we're a startup
- It's in beta, users will find the bugs
- We don't have enough money
- We haven't made any major changes
- I didn't think hackers would target us
- They're using it wrong
- http://blog.utest.com/8-common-excuses-in-software-testing/2013/01/

Praxe: vývoj browseru Chromium (Chrome)

- Používají se unit testy, testy API, UI testy, výkonnostní testy, bezpečnostní testy, atd.
- Součástí zdrojových kódů jsou základní unit, UI a další testy, které by vývojáři měli používat při vývoji
- Každé potvrzení (*commit*) do repozitáře spustí automatické sestavování kódu pro jednotlivé platformy a nastavení (building builders) a následné testování, pokud bylo sestavení úspěšné (testing builders)
- Testy jsou specifické pro jednotlivé platformy
- Zabraňuje se tak regresím a zvyšuje se produktivita
- Testy jsou zde integrální součástí procesu vývoje
- https://sites.google.com/a/chromium.org/dev/developers/testing

Chromium: Console



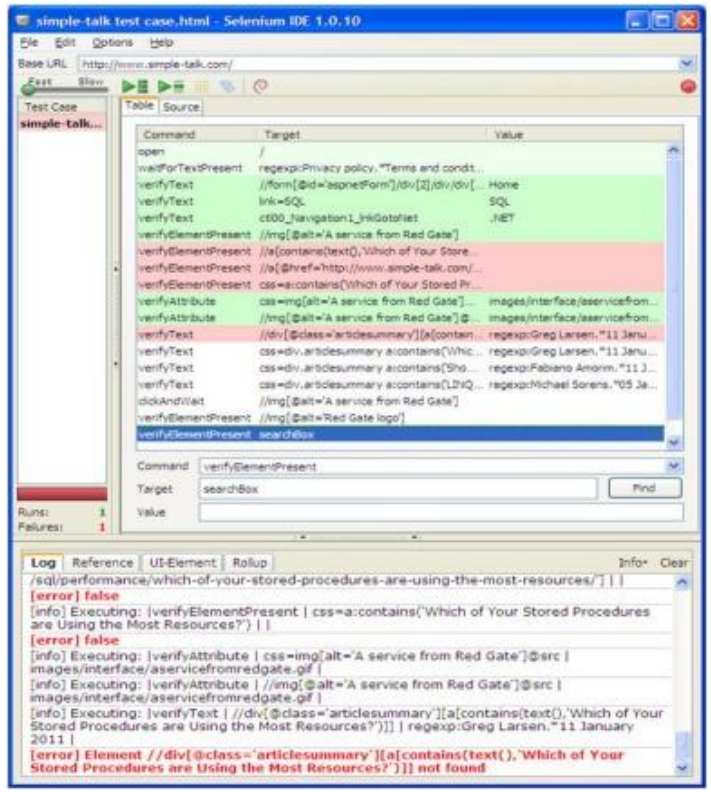
http://build.chromium.org/p/chromium/console

Chromium: Waterfall

http://build.chromium.org/p/chromium/waterfall

Něco navíc: testování GUI aplikací

- Funkční testování
- Selenium
  - Automatizace ve webových prohlížečích
  - Java API
  - Chrome, FireFOx, IE, Safari, Opera



Zdroje

- [1] I. Burnstein: Practical Software Testing, Springer-Verlag. 2003.
- [2] K. Beck: Simple Smalltalk Testing: With Patterns. 1989. http://www.xprogramming.com/testfram.htm
- [3] http://behaviour-driven.org/Implementations
- [4] M. Fowler: Mocks aren't stubs
- http://martinfowler.com/articles/mocksArentStubs.html
- [5] S. Ducassé: SUnit. http://stephane.ducasse.free.fr/Resources/LecturesInPowerpoint/STOOP- 450-SUnit.ppt
- [6] M. Fowler: Continuous Integration. http://www.martinfowler.com/articles/continuousIntegration.html
- [7] Putting It to the Test. http://blog.chromium.org/2008/11/putting-it-to-test.html