

IV113 Validace a verifikace

Techniky Testování II

doc RNDr. Jiří Barnat, Ph.D.

Funkční testování

Princip techniky

- Oddělené testování malých částí kódu na úrovni vnitřních rozhraní (API).
- Testování jednotlivých funkcí bez vnitřní znalosti jejich implementace (black-box testing).
- Vyžaduje modulární kód.

Základní cíl

- Provéřit, že izolované funkce systému fungují správně.
- Odladěná podřešení se lépe kombinují do funkčního celku.

Globální postup

- Identifikují se vnitřní funkce produktu.
- Pro každou identifikovanou funkci se vytvoří test.

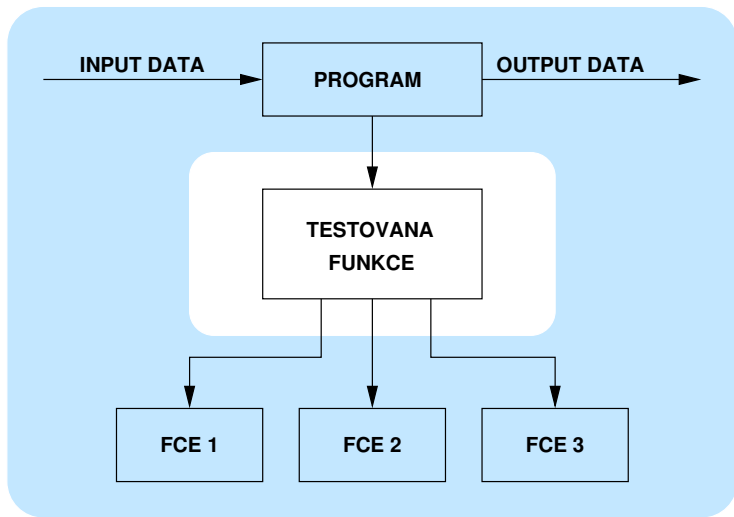
Testování izolované funkce

- Funkce izolovaná od systému není samostatně spustitelná.
- Je třeba vyvinout software, který umožní testování funkce.
- **Implementace testovacího prostředí probíhá jako součást implementace funkce.**

Testovací prostředí

- Vzhledem k testované funkci má vnější a vnitřní část.
- **Vnější část** slouží jako hlavní, samostatně spustitelný program, který volá testovanou funkci s danými parametry, sbírá a tiskne relevantní výsledky volání funkce.
- **Vnitřní část** simuluje chování dalších funkcí volaných testovanou funkcí.

Obecné schéma testovací prostředí funkce



Podpora pro funkční testování

- **Java:** junit, TestNG, qc4j
- **C++:** CUnit, TUT, libwibble, QuickCheck++
- **Haskell:** QuickCheck, SmallCheck, HUnit
- **Python:** PyUnit, nose, qc
- **Ruby:** Test::Unit, RushCheck
- ...

Funkce

- Funkce je něco, co program může dělat.
- “Features”, schopnosti, entity identifikované svými schopnostmi, ...

Identifikace funkcí

- Ze specifikace, či manuálu.
- Z uživatelského rozhraní.
- Z nápovědy v GUI/TUI.
- Prohledáním zdrojového kódu (názvy členských funkcí tříd, texty chybových hlášek, ...).

Seznam funkcí

- Základní dokument funkčního testování.

Informace obsažené v seznamu funkcí

- Kategorizace funkce, tj. označení skupiny funkcí s podobnou, či související funkcionalitou.
- Vstupy funkce
 - Maximum/Minimum, hraniční případy.
 - Speciální případy
- Výstupy funkce
- Rozsah působnosti funkce (není-li dána kategorizací).
- Možnosti/volby (options) funkce.
- Okolnosti, za kterých se funkce chová odlišně (globální konfigurace programu, verze a typ OS, ...)

Orákulum

- Je nutné vědět (nebo mít určeno) jak se pozná, že daný test dané funkce uspěl.
- Orákulum, může být součástí seznamu funkcí.

Neúplnost testování

- Není-li možné otestovat všechny vstupy, je vhodné použít princip doménového testování.

Konfigurace systému a vliv prostředí

- Testy funkce je vhodné opakovat v potenciálně různých podmínkách, při nichž je možné funkci využít.

Testování negativních případů

- Funkce by se měla testovat na to, že dělá to, co dělat má, ale i na to, že nedělá to, co dělat nemá.

Pokrytí (coverage)

- Technika vhodná k realizaci úplného pokrytí testy.
- Vhodná technika k budování testovacího plánu, potažmo k měření míry splnění testovacího plánu.

Ranné testování

- Lze testovat už částečně vzniklý kód.
- Vede k rychlému odhalení mnoha chyb.

Rizika použití funkčního testování

- Pokud je v projektu přítomno, ochabuje potřeba realizovat jiné metody testování.
- Produkt, který je vystavěn z korektních funkcí, nemusí být korektní.

Nedostatky funkčního testování

- Neuvažuje interakci jednotlivých funkcí na stejné úrovni.
- S rostoucí integrací funkcí "ztrácí sílu".
- Nezachycuje chování funkcí v dlouhodobém běhu.
- Často se zaměřuje na testování schopnosti jako takové, ale ne na testování krajních případů.
- Neřeší otázky typu: Byla funkcí produktu naplněna uživatelská potřeba?

Testování odvozené od možných rizik Risk-based testing

Pozorování

- Jednotlivé dosud probrané metody testování se příliš zaměřují na jednotlivé aspekty/způsoby použití produktu.
- Důkladná realizace jedné z metod se postupem času stává nákladná a neúčinná v detekci jiných nedostatků produktu.

Pragmatický pohled

- Jednotlivé testy předepsané důkladnou aplikací daných testovacích metod je třeba podrobit analýze a zvážit, zda jejich provedení pomůže naplnit misi.
- Volbu správné strategie, metody testování a realizovaných testů je možné provést na **základě možných rizik**.

Riziko

- Možnost utrpět ztrátu či být postižen nějakou škodou.

Dimenze rizika v softwarovém inženýrství

- Jakým způsobem může program selhat.
- Jak pravděpodobné je, že program selže.
- Jaké jsou důsledky selhání programu.

Princip techniky testování rizikem

- Uvědomění si, jakým způsobem může program selhat.
- Návrh testů, které odhalí myšlené (potencionální) selhání.

Testování odvozené od možných rizik

- Přirozeně (nevědomky) používaná metoda, často již samotnými vývojáři systému.
- Typická metoda pro hledání chyb.
- Snadno se prolíná s ostatními metodami.

Příklad – využití rizika v doménovém testování

- V číselných doménách je často používána 0 jako jeden z hraničních případů. Důvodem je mimo jiné i to, že existuje riziko dělení nulou.
- I na volbu hraničních případů lze nahlížet jako na volbu zdůvodněnou rizikem záměny $<$ a \leq .

Otázka

- Jakým způsobem může program selhat?

Hledání odpovědi

- Mnoha způsoby, úplného výčtu není možné efektivně dosáhnout.
- Pro identifikaci rizik se používají zejména zkušenosti z předchozích projektů, a různé heuristiky.

Správa projektu

- Nové části produktu
 - Riziko dosud neodhalených chyb.
- Nová technologie v produktu
 - Riziko dosud neprojevených chyb.
- Proces učení
 - V učícím procesu lidé dělají více i nestandardních chyb.
- Modifikované části
 - Riziko nově zanesených chyb.
- Změny na poslední chvíli
 - Riziko nesouladu se zbytkem projektu.
- Části vyvíjené v časové/finanční tísní
 - Nezralost kódu, zvýšená pravděpodobnost chyb.

Lidský faktor

- Vyčerpanost
 - Unavení vývojáři dělají více chyb.
- Distribuovaný tým
 - Riziko nedostatečné komunikace, potažmo nekonzistence v kódu.
- Osobní problémy
 - Nesoustředěný vývojář vyprodukuje kód s větším množstvím chyb.
- Nečekané “features”
 - Riziko nekonzistence se zbývajících částí produktu.
- Práce odvedená třetí stranou
 - Třetí strana nemusí mít dobré pochopení lokálního kódu.
 - Důvěřuj, ale prověřuj.
- Práce odvedená zdarma
 - Riziko neúplnosti kódu.

Specifikace a požadavky

- Nepřesná specifikace
 - Riziko různé interpretace / odlišného zúplnění.
- Konfliktní požadavky
 - Riziko, že vyřešením konfliktu se odstraní část požadované funkcionality.
- Záhadná mlčenlivost
 - Riziko, že mlčení zakrývá problémové místo v programu.
- Požadavky vyvíjející se za běhu
 - Riziko, že změny požadavků nejsou zpracovány.

Výskyt chyb

- Chybové části
 - Riziko neodstranění všech chyb. Kde bylo mnoho chyb, je pravděpodobné, že budou další.
- Nedávno chybové části
 - Riziko dosud neodhalené nedokonalosti nápravy.
- Silné závislosti
 - Riziko, že případná chyba v této části kódu ovlivní celý produkt.
- Distribuce
 - Cokoliv co je distribuováno v čase či prostoru a má fungovat jako celek je náchylné na chyby.
- Komplexní části
 - Těžko pochopitelné je i těžko programovatelné.
- Typicky chybové konstrukty jazyka
 - Pointery v C.

Proces testování

- Málo testované části
 - Riziko dosud neodhalené chyby.
- Nedostatečná různorodost testovacích technik
 - Riziko systematického míjení chyby.
- Slabé testovací nástroje
 - Bez použití nástroje jako valgrind je pravděpodobnost výskytu chyby související s přístupem do neoprávněné části paměti mnohem vyšší.
- Neopravitelné chyby
 - Chyby, které nebylo možné opravit v okamžiku jejich nahlášení nesou riziko nedostatečné či nesprávné opravy.
- Netestovatelné části
 - Části kódu, které vyžadují pomalé, obtížné či jinak neefektivní testování, jsou často nedostatečně testovány.

Prevence

- Testování částí, které v případě projevu chyby mohou mít důsledky.

Možné důsledky projevení chyby

- Špatná publicita,
- právním následky,
- výrazné škody,
- nesoulad s požadavky,
- zneužití produktu,
- zklamání většiny uživatelů,
- ohrožení strategického postavení,
- neuspokojení V.I.P. osob, . . .

Katalog chyb

- Veřejné seznamy nejčastěji se vyskytujících chyb.
- Seznam možných způsobů selhání produktu.
- http://www.logigear.com/logi_media_dir/Documents/tcs_appA_bugs.pdf

Použití katalogu

- Generování testů.
 - Pro konkrétní chybu v seznamu se ptám:
 - Může chyba v testovaném produktu nastat?
 - Je chyba smysluplná a jaké jsou případné následky?
 - Jakým způsobem je možné chybu detekovat?
 - Vytvořím test, který produkt na chybu prověří.
- Audit testovacího plánu.
- Identifikace nových/nestandardních způsobů testování.
- Způsob zaučení nových testerů.

Instalační program

- Instalace špatných souborů
 - Neodstranění dočasných souborů.
 - Neodstranění zastaralých souborů.
 - Instalace nepotřebných souborů.
 - Neinstalace potřebných souborů.
 - Instalace do nesprávného adresáře.
- Poškození souborů
 - Starší verze nahradí novější verzi.
 - Data uživatelů poškozena během povýšení verze.
- Poškození jiných aplikací
 - Modifikace sdílených souborů.
 - Smazání souborů využívaných jinou aplikací.
- ...

Listování produktem po jednotlivých komponentách.

- Identifikace komponent
 - struktura, funkcionalita, data
 - způsob použití, cílové platformy
- Identifikace způsobů, jakým je možné, aby daná komponenta mohla selhat (bez ohledu na to, jak je realizována).
- Identifikace důvodů selhání, průvodních jevů a případných dopadů.
- Vyplatí se, aby byl do testovacího plánu vložen daný test?

Katalogy chyb

- Nejsou ortogonální (netriviální překryvy).
- Jsou nevhodné pro klasifikaci použitých technik/myšlenek.

Dimenze rizika v softwarovém inženýrství

- Jakým způsobem může program selhat.
- **Jak pravděpodobné je, že program selže.**
- **Jaké jsou důsledky selhání programu.**

Odhad míry rizika

- Ohodnocení pravděpodobnosti výskytu a závažnosti důsledků číslem v rozmezí [1-10]
- Míra rizika = [Pravděpodobnost] x [Důsledky]

Management

- Preference odstranění chyb s větší mírou rizika.

Nedostatky

- Jaká je pravděpodobnost výskytu chyby v produktu?
- Jaký je rozdíl mezi známkou 3, 4 a 5?

Příklad – Borland's Turbo C++

- Chyba: poškození projektového souboru
- Pravděpodobnost: 1 (very rare)
- Závažnost: 10 (critical)
- Míra rizika uvedené chyby: 10
- Nejzávažnější chyba projektu, ale v měřítku míry rizika dosahuje pouze 26% maxima.
- Chyba [5]x[5] je o 50% závažnější.

Testovací cyklus rizikového testování

- Analýza selhání
- Zdokonalení procesu analýzy
- Určení priorit jednotlivých rizik
- Provedení odpovídajících testů
- Ohlášení chyb
- Náprava některých problémů

Regresní testování

Princip techniky

- Opakování vybrané sady úspěšně proběhnuvších testů.

Hlavní důvod použití

- Riziko zanesení chyb změnou kódu.

Další uplatnění

- Potvrzení stability chování/výkonu produktu.
- Nástroj pro prokázání množství odvedené práce klientům.
- Psychologická podpora vývojářů.
(Vývojáři mohou být odvážnější při změnách kódu.)

Oprava chyby je nedostatečná.

- Záplata je neúčinná
- Záplata odstraní symptomy, ne však chybu samotnou.

Oprava chyby má vedlejší účinky.

- Výskyt nově zanesených chyb.
- Znovu vyvolání opravených chyb.

Produkt nelze sestavit.

- Typicky ve spojení se systémem pro kontrolu verzí.

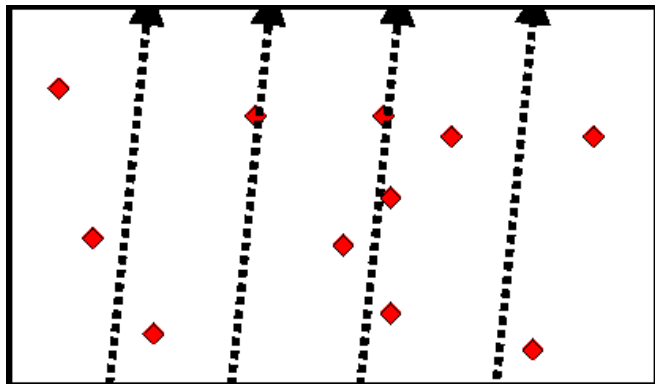
Opakování testů za účelem odhalení nových chyb.

- Produkt se chová deterministicky.
- Použitím stejných testů neodhalíme mnoho nových chyb.
- Může odhalit chyby, které se projevují jen někdy:
 - chyby závislé na předchozím použití produktu,
 - chyby, jež se projeví díky přítomnosti nového kódu,
 - ...

Opakování stejných testů

- Opakování silných testů, tj. takových testů, kterými pokud testovaný produkt projde jednou, projde jimi pravděpodobně kdykoliv, může být bezpředmětné.
- Opakování slabých (úzce zaměřených) testů může být, co do počtu odhalení chyb, úspěšnější.

Příklad – Analogie s minovým polem



Podstata regresního testování je v opakování testů.

Ptáme se:

- Které testy mají být součástí sady?
- Jaký je důvod pro opakování právě těchto testů?
- Jak přesně se mají jednotlivé testy v sadě vyhodnocovat?

Pozorování

- Podobně jako testování na základě rizika, i regresní testování je jde napříč předchozích technik testování.

Procedurální

- Opakujeme vybranou, nadále stejnou, sadu stejně vyhodnocovaných testů stejným způsobem.

Ekonomické

- Opakujeme všechny snadno opakovatelné a vyhodnotitelné testy.

Zaměřené na snížení rizika

- Opakujeme existující testy, jež v minulosti odhalily chyby, a volíme testy, jež pokrývají kritické části produktu.

Podpora při vývoji produktu

- Volíme testy, které pomohou rozhodnout, zda je korektní/smysluplné modifikovat produkt navrženým způsobem (sledujeme např. výkon aplikace).

Pozorování

- Produkt se vyvíjí a spolu s ním je nutné vyvíjet i testy, které se mají znovu spustit.
- Udržovat testy v aktuálním spustitelném stavu může být nákladné.
- **Nebrání údržba starých testů ve vývoji nových testů pro dosud neotestované části produktu?**

Obecný princip

- Vytvoření nových účinnějších testů s použitím existujících.
- Znovupoužití práce související s přípravou testů.
- Typicky v průběhu testovacího procesu dochází k testování dané funkcionality produktu pomocí několikrát zdokonalených testů, tj. testů s větší silou odhalit možné chyby.

Způsoby zesílení testů

- Více iterací stejného testu.
 - Snižuje riziko zdánlivě nedeterministické chyby.
- Kombinace několika testů do jednoho testu.
- Rozšiřování testů.
 - Zahrnutí nových "features".
 - Zahrnutí podčástí, ve kterých se objevila chyba.

Pozorování

- Pokud je programátor sám schopen provést testy kódu, má možnost případné chyby způsobené změnou kódu ihned odhalit a odstranit bez vytváření záznamu v systému pro správu chyb.
- V opačném případě (chybu ihned neodhalí) proběhne netriviální proces, než se programátor dostane k tomu, aby chybu opravil.

Pozorování

- Regresní testování se typicky realizuje pomocí funkčního testování (unit testing).

Důvody

- Ve větších projektech je typické, že programátor samotný tvoří testy pro část kódu, který vyvíjí.
- Automatizovatelné od raného stádia vývoje.

Rizika

- Po dokončení vývoje modulů, metoda postrádá smysl.
- Sadu testů je nutné obohacovat o integrační testy.
 - Unit testy si tvoří sami vývojáři, jakmile však dochází k integraci, je pro vytvoření testu nutná znalost všech integrovaných částí, což může být na rámec působnosti každého jednotlivého vývojáře.

Další techniky testování

- Funkční testování
- Doménové testování
- Testování na základě rizika
- Scénářové testování
- Regresní testování
- Fuzz testování
- Testování vůči specifikaci (Specification-based)
- Testování zátěží (Stress)
- Testování uživatelem (User)
- Testování s použitím modelu (Model-based)

Princip

- Tvorba nových testů náhodnou modifikací (mutační testování) nebo vypočtenou modifikací (whitebox fuzz testování) vstupních dat existujících testů.
- Genetické algoritmy pro tvorbu nových testů.

Motivace

- V black-box variantě levná a překvapivě úspěšná metoda.
- Docílit průchod jiných částí kódu (zvýšit pokrytí).

Problémy

- Ve white-box variantě je systematický výpočet modifikace vstupních dat náročný.

Princip

- Testování jednotlivých tvrzení ve specifikaci.

Motivace

- Chceme soulad se specifikací.
- Prokážeme souladu se specifikací.
- Metoda správy testování a testovacího úsilí jako celku.

Problémy

- Nevyjádřená, implicitní a nejasná fakta ve specifikaci.
- Zaměřeno na pokrytí spíše než na eliminaci možných rizik.

Princip

- Pozorovat chování produktu při enormním zatížení.
- Testy mají za úkol vynutit chybu v produktu způsobenou zahlcením produktu.

Motivace

- Získání důvěry v chování produktu v kritických momentech.
- Prevence odhalení nových chyb masovým používáním produktu uživateli.

Problémy

- Generování odpovídající zátěže.

Princip

- Simulace uvolnění produktu.
- Uvolnění produktu omezené skupině uživatelů, kteří mají za úkol ověřit funkčnost produktu.
- Beta test.

Motivace

- Kvalitu posuzují sami cíloví zákazníci. Ti odhalí skutečné problémy produktu.

Problémy

- Vyžaduje téměř finální verzi produktu.
- Získávání beta testerů.

Princip

- Modelování produktu jako konečného automatu.
- Odvozování vlastností a nutné množiny testů na základě modelu.

Motivace

- Přiblížení se formální verifikaci.
- Matematická garance vlastností modelu potažmo produktu samotného.
- Generování minimální množiny testů.

Problémy

- Náročnost budování věrného modelu.

Skriptování

Skript

- Posloupnost vstupních a výstupních akcí spojených s provedením testu.
- Definice testu popsáním procedury, která vede k ověření testovaných vlastností produktu.

Motivace

- Skript je natolik jednoduchý a přesný, že jeho vykonávání je čistě mechanické a může být provedeno robotem.
- Testy specifikované pomocí skriptů, je možné nechat provést počítačem či lidmi s nízkou kvalifikací v testování.

Příklad

- 1) Rozbal **File** menu
- 2) Vyber volbu **Print**
- 3) Program zobrazí **Dialogové okno s možnostmi tisku**
- 4) Vlož **2** do políčka **Počet kopií**
- ...

Možná forma

#	Akce	Očekávaná reakce	Komentář	Odchylka
1	Rozbal File menu	File menu rozbaleno	Začátek testu	e
2		
3		

Opakovatelnost

- Různí testeři sledující tentýž skript provedou stejné akce a dosáhnou stejných výsledků testu.

Zajištění odpovídající kvality testování

- Vykonávání skriptovaného testu zajistí vykonání všech naplánovaných akcí v rámci testu.

Využití různé úrovně schopností testerů

- Testeři senioři vytvoří skript.
- Testeři junioři skripty provádějí.
- Nejlepší lidé v testovací skupině mají možnost věnovat se skutečným problémům spojeným s testováním produktu.
- Skripty jsou důležité a musí je vytvořit testeři, jejich vykonání však může provést jiná skupina v rámci projektu.
- Možný způsob trénování nových zaměstnanců na projektu.
- Možnost nábory nových zaměstnanců.

Příklad

- U pramene vody jsou k dispozici tři různě velké nádoby. Úkolem je pomocí těchto nádob naměřit určené množství vody.

Problém	Velikosti nádob			Cílové množství
1	21	127	3	100
2				
3				
4				
5				
6				

Příklad

- U pramene vody jsou k dispozici tři různě velké nádoby. Úkolem je pomocí těchto nádob naměřit určené množství vody.

Problém	Velikosti nádob			Cílové množství
1	21	127	3	100
2	14	163	25	99
3				
4				
5				
6				

Příklad

- U pramene vody jsou k dispozici tři různě velké nádoby. Úkolem je pomocí těchto nádob naměřit určené množství vody.

Problém	Velikosti nádob			Cílové množství
1	21	127	3	100
2	14	163	25	99
3	9	142	6	121
4				
5				
6				

Příklad

- U pramene vody jsou k dispozici tři různě velké nádoby. Úkolem je pomocí těchto nádob naměřit určené množství vody.

Problém	Velikosti nádob			Cílové množství
1	21	127	3	100
2	14	163	25	99
3	9	142	6	121
4	18	43	10	5
5				
6				

Příklad

- U pramene vody jsou k dispozici tři různě velké nádoby. Úkolem je pomocí těchto nádob naměřit určené množství vody.

Problém	Velikosti nádob			Cílové množství
1	21	127	3	100
2	14	163	25	99
3	9	142	6	121
4	18	43	10	5
5	22	47	3	19
6				

Příklad

- U pramene vody jsou k dispozici tři různě velké nádoby. Úkolem je pomocí těchto nádob naměřit určené množství vody.

Problém	Velikosti nádob			Cílové množství
1	21	127	3	100
2	14	163	25	99
3	9	142	6	121
4	18	43	10	5
5	22	47	3	19
6	23	49	3	20

Pozorování

- Lidé mají tendenci postupovat podle osvědčených a naučených postupů.
- Jakmile máme naučen jistý postup, nevědomky ignorujeme mnohé ač relevantní informace.

Přehlízíme, že

- Jiná procedura, by mohla být efektivnější.
- Aplikujeme proceduru na podobnou avšak nekompatibilní situaci.
- Se stalo něco relevantního, co by mohlo ovlivnit to, co je výsledkem aplikované procedury, ale není to v rozsahu vnímání předepsaného danou procedurou.

Předpoklad

- Jestliže někdo postupuje podle předem dané jednoduché posloupnosti příkazů, má oči otevřené a všimne si neobvyklých věcí, drobných odchylek, nesouladu s popisovaným výstupem, atd.

Další příklady – inattentional blindness

- Změna dopravního značení na 100x projetém úseku.
- Visual Congition Lab

Pozorování

- Nejvíce chyb se odhalí při tvorbě skriptů.
- Učení prováděním skriptů má po skončení počáteční seznamovací fáze výrazný pokles v účinnosti.
- Je možné použít skripty k identifikaci zajímavých míst, které je vhodné otestovat jiným způsobem. (Nedůsledné provádění instrukcí skriptu.)
- Význam skriptování je spíše ve strukturovaném zkoumání produktu.
- Drahé a neúčinné.

Automatizace

Motivace

- Část testovací práce odvede počítač.

Problémy

- Implementace orákula (viz nevýhody skriptování).
- Jak velkou část práce je schopna automatizace ušetřit?

- Analýza produktu – člověk
- Návrh testů – člověk
- Spuštění iniciálních testů – člověk
- Vyhodnocení výsledků – člověk
- Hlášení chyb – člověk
- Ukládání verzí, kódů s chybou – člověk
- Dokumentace testovacího procesu – člověk
- Znovu-spuštění testů – **stroj**
- Vyhodnocení výsledků – **stroj**+člověk
- Udržování výsledků – člověk

Automatizovaná procedura testování

- “Jedním příkazem” se spustí sada testů, ty se provedou, vyhodnotí a na výstupu se zobrazí statistiky, případně identifikují neúspěšné testy.

Autonomní procedura testování

- Spouští se bez explicitního příkazu testera/uživatele.
- Spouštěcí mechanismy
 - časová periodičita (každou půlnoc)
 - událostmi řízené spouštění
(commit do systému pro správu verzí)

Připomenutí

- Znovu-spuštění testů – **stroj**
- Vyhodnocení výsledků – **stroj**+člověk

⇒ Regresní testování

BuildBot

- Systém pro podporu automatické kompilace a testování.
- Umožňuje spouštění testů na různých platformách.
- `buildbot.net`

Jiná řešení

- `cdash`, `tinderbox`, ...

C/C++, Linux

- Nástroje gcov and lcov.

Příklad: lcov

- gcc -fprofile-arcs -ftest-coverage foo.c -o foo
lcov -d . -z
lcov -c -i -d . -o base.info
./foo
lcov -c -d . -o collect.info
lcov -d . -a base.info -a collect.info -o result.info
genhtml result.info