Úvod do Software Quality Assurance

Co, proč a Jak

Co je to software QA?

Oficiální definice

- *„Procedura podrobující předmět testu takovým podmínkám či operacím, které vedou k jeho přijetí či odmítnutí”* **(Merriam-Webster)**

- *„Proces zahrnující plánování, přípravu a hodnocení SW s cílem zjistit, zda splňuje stanovené požadavky; prokázat, že je vhodný pro svůj účel; a najít defekty”* **(ISTQB)**

Samé otázky, více otázek

- *„Podrobit podmínkám”*?
  Jakým? Jak to zjistit?
- *„Zda splňuje požadavky”*?
  Jaké? Kde je vzít?
- *„Zda je vhodný pro svůj účel”*?
  Jak se to prokazuje?
- *„Najít defekty”*?
  Kolik stačí? Jak je hledat? Mají se počítat?
- „Testování“ je proces podrobování systému *zkouškám*...
- Ale jak zjistíme *jaké* zkoušky vůbec chceme provádět? A *jak* je provádět?

Udělejme si v tom jasno

- **Quality Control versus Quality Assurance**
  - QC: Reaktivní – kontrola po dokončení vývoje (testování, code reviews...)
  - QA: Preventivní – zpětná vazba napříč lifecycle (analytika, architektura, procesy, review, QC, RCA)
- **Defekt, Chyba, Selhání**
  - Defekt: objektivní vada systému
  - Chyba: následek defektu který se propaguje systémem
  - Selhání: nesprávné chování komponentu či systému v důsledku defektu nebo chyby
- **Verifikace versus Validace**
  - Verifikace: Stavíme to správně (=nejsou defekty v analýze, kódu...)
  - Validace: Splňujeme požadavky zákazníka (=chápeme správně co máme stavět)
- **Tester versus Test analytik**
  - Tester: vykonává či spouští a vyhodnocuje testovací scénáře
  - Test analytik: systematickou analýzou určí potenciální selhání a připraví scénáře které je pokryjí

SW QA trpí strašlivým zmatením pojmů a dojmů

„Slovo, které může znamenat cokoli, neznamená nakonec nic“

- Tester ≠ QA
- Tester ≠ Test analytik
- Kvalita se magicky nezlepší tím, že QC a testerům dáte vznešenější názvy
  - Test analytik nevznikne přejmenováním, ale nabytím technik a dovedností pro TA
  - Quality Assurance nevznikne přejmenováním, ale praktickým přijetím procesů a nástrojů pro QA
- Kvalita se magicky nezlepší tím, že zavedete procesy a ISO9001 „Quality management“
  - Proces je pouze nástroj, který umožňuje dosáhnout nějakého cíle
  - Proces musí být reálně užitečný, nebýt jen formalitou/byrokracií (dobrý sluha, zlý pán)
  - Lidé musí chtít a umět procesy využít

Taxinomie má ve skutečnosti smysl

- **Statické versus Dynamické metody**
  - Statické: SUT se nepouští, ale dělá se analýza (kódu, architektury, stavů, prostředí...)
  - Dynamické: SUT se pouští a je pozorováno zda výstupy odpovídají daným vstupům
- **Black-box versus White-box testování**
  - White-box: se znalostí kódu (=jediná detekce některých chyb X ovlivnění předpoklady tvůrce)
  - Black-box: bez znalosti kódu (=jediné skutečně nezávislé ověření X nevidí do interních věcí)
- **Jednotkové versus Systémové versus Integrační testy**
  - Jednotkové: verifikace funkce nejmenších jednotek (procedur, funkcí, tříd, makra)
  - Systémové: verifikace a validace funkce
  - Validace: Splňujeme požadavky zákazníka (=chápeme správně co máme stavět)

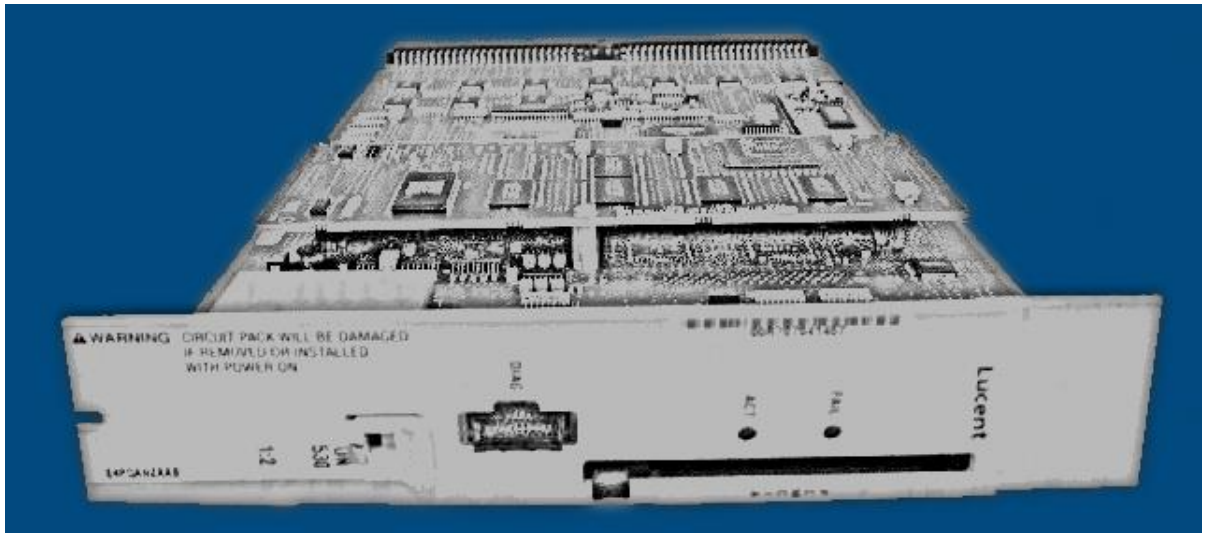
Druhy defektů → co je umí odhalit

1. **Chyby kódu**
   a) Chyby syntaxe → CASE (validátory syntaxe)
   b) Chyby v logice → Statické White-box testy (Unit testy, Code Review)
   c) Race conditions → CASE (nástroje statické analýzy kódu)
2. **Chyby implementace**
   a) Špatné chápání požadavků → Systémové/Akceptační testování
      → V&V metodologie vývoje (Scrum, Spiral, V-Model)
   b) Nesprávné předpoklady → Nezávislé Black-box testy po důkladné Test analýze
      → Stress testy, Load testy
   c) Neošetřené stavy → State Transition Testing, BVT
   d) Vnější faktory → Safety/Reliability analýza (FMEA, FTA)
      → Integrační testování
      → Environmentální testování

Proč potřebujeme software QA?

Bell 4ESS, výpadek celé N.Y. sítě AT&T, 15. ledna 1990

- Update FW-digitální Switche v ústřednách AT&T
- Když se switch restartoval a vyslal 1. zprávu, ostatní switche provedly interní reset že už je online
- Když v průběhu tohoto resetu dostaly 2. zprávu z restartovaného switche, spadly
- Po restartu vyslaly 1. zprávu, ostatní switche začaly provádět interní reset, když dostaly 2...
- Rekurzní chyba, všech 114 páteřních switchů se restartovalo každých 6 sekund
- Výpadek 9 hodin v kuse
- (Workaround: snížení propustnosti komunikačního kanálu, poté downgrade FW)
- **Failure Mode testing && Integration testing**

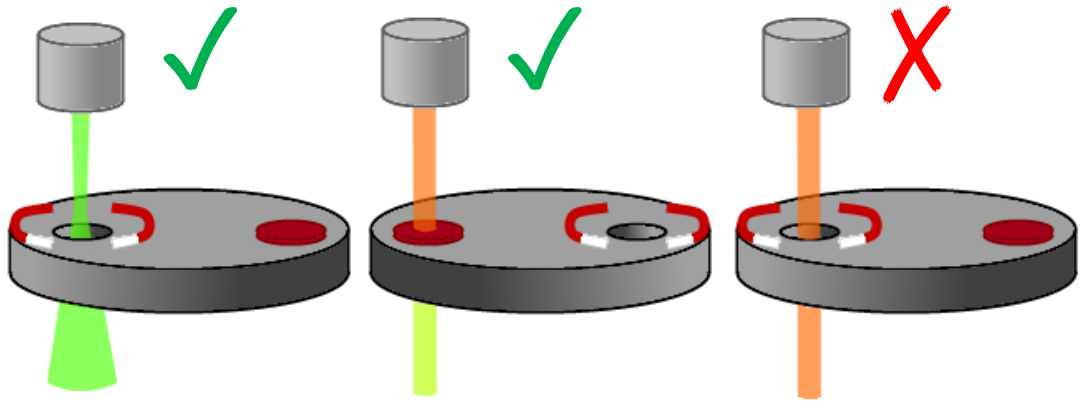


http://phworld.org/history/attcrash.htm

ca technologies

Therac-25, Ozáření pacientů, 1985-1987 (3 mrtví, další měli nemoc z ozáření)

- Přístroj pro radioterapii měl 3 režimy:
  1. přímé ozařování elektrony s nízkou energií s magneticky řízeným svazkem
  2. ozařování 100× vyšší energií přes vložný štít (pro konverzi proudu elektronů na Rtg záření)
  3. zaměření světelným terčem bez vychylování
- Therac-25 nahradil mechanické pojistky SW
- Žádné ověření správného fungování senzorů
- Nebyly ošetřeny race conditions
  - pokud operátor vybral mód 2) a vzápětí přepnul na mód 1), došlo k ozařování pacienta 100× vyšší energií pro mód 2) bez vložení kovového štítu
  - v módu 3) se mohl aktivovat mód 1), ale fixně bez EM vychylování, takže spálil 1 místo na těle
- **Statická inspekce; State transition testing**



http://radonc.wikidot.com/radiation-accident-therac25

Ariane 5G, exploze letu 501, 4. června 1996 (370 milionů USD)

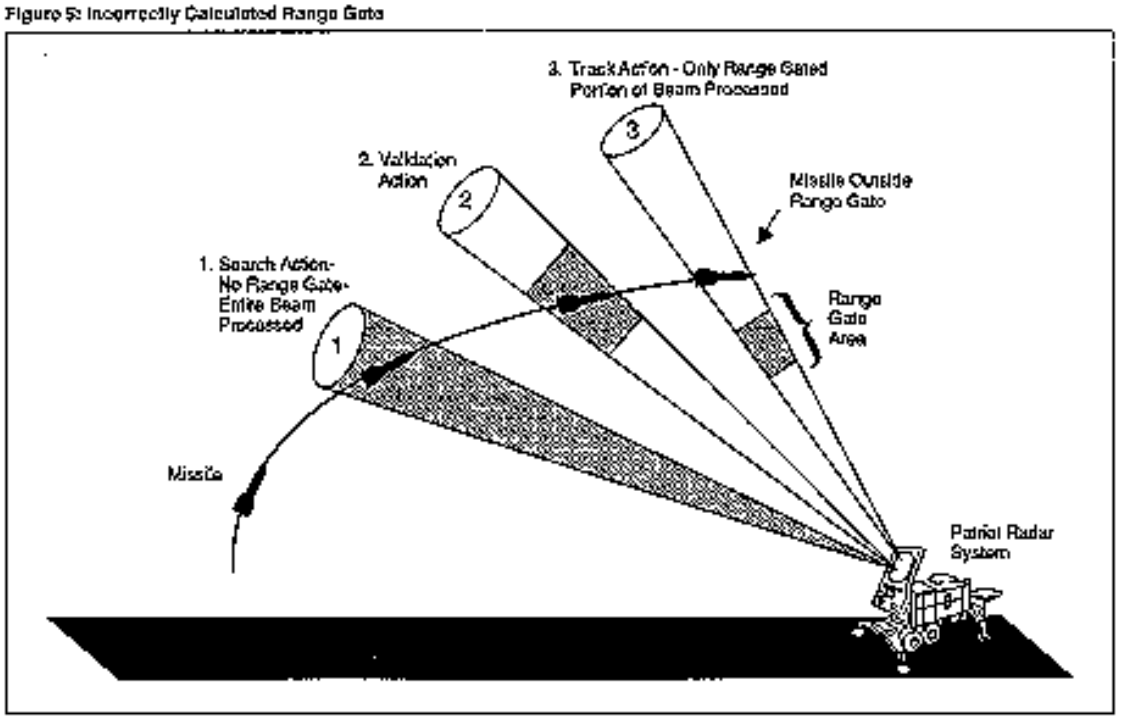
- Defekt: konverze z 64b float na 16b sint; hodnoty naměřené Ariane 5 se nevešly do 16b → overflow
- ADA detekuje overflow, brání jim
- Chyba: kód zahlásil OPERAND_ERROR, SRI vypsal binární diagnostický dump paměti
- Paralelní selhání PRI i BACKUP: stejný SW; „hot“
- Selhání: dump byl interpretován jako letová data!
- Systém vyhodnotil „data“ jako extrémní odchylku, zkusil ji kompenzovat zatáčkou o > 20°
- Aerodynamické síly vedly k roztržení rakety a LOV
- Tento kód zbytečně běžel 40 sekund po startu A5G
- **FTA s CCF, Stress testing, FMT v rámci Integ.T, <u>Boundary Values Testing pro A5G</u>**

http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html

ca technologies

MIM-104 Patriot, selhání PRO, 25. února 1991 (28 mrtvých, 100 zraněných)

- Systém protiraketové obrany Patriot nesestřelil útočící iráckou BŘS Scud kvůli selhání SW
- Scud se ztratil z radaru Patriotu protože SW špatně určil, jak daleko má radar hledat cíl
- Vzdálenost radaru pro vyhledávání byla funkcí
  - rychlosti cíle
  - sekund od startu počítače Patriotu (system clock)
- Problém: chyba v zaokrouhlení přičítala offset 0,000000095d každý tick (1/10 sek.)
- Důsledek: chyba v určení vzdálenosti += 7 metrů každou hodinu, po 20 hodinách ztráta zaměření
- Střelecká baterie A běžela nepřetržitě už 100 hodin, nezaměřila Scud, ten zasáhl kasárna
- **Stress testing, Boundary values testing?**



GAO/IMTEC-92-26, used under fair-use

http://www-users.math.umn.edu/~arnold/disasters/patriot.html

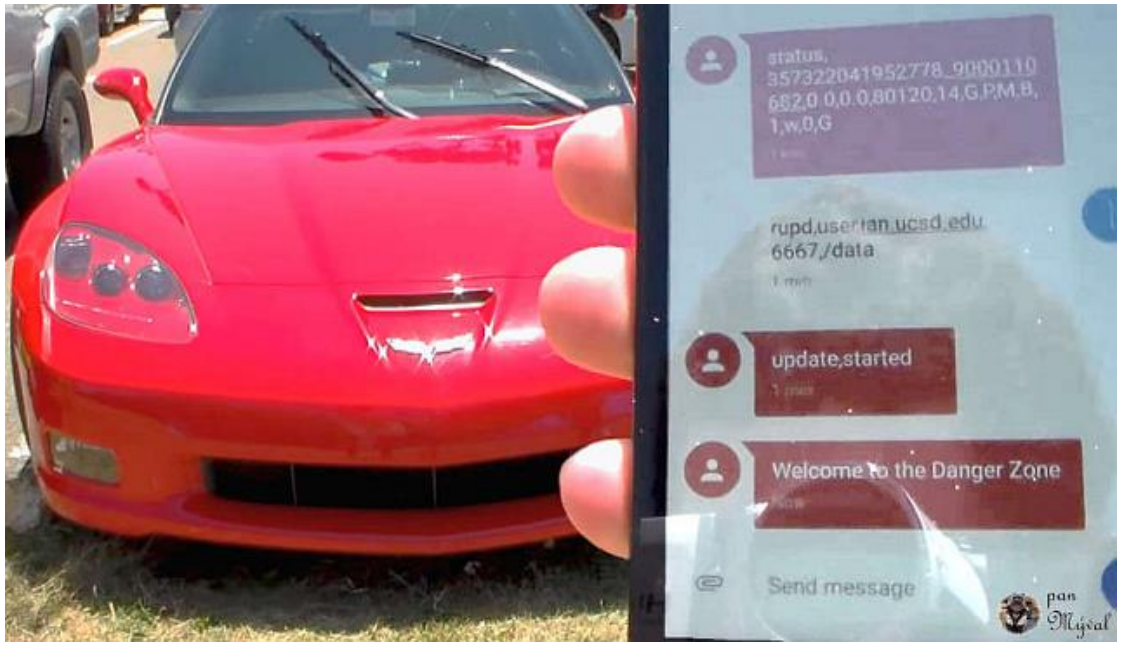
Airbus 330-203, ztráta vztlaku a LOV, 1. června 2009 (všech 228 osob mrtvých)

- Posádka v nulové viditelnosti „přetáhla“ letoun, došlo ke ztrátě vztlaku, propadu, *UFIT*
- Vedlejší příčina: převrácená signalizace „STALL“ mátla pilota když chtěl přestat přitahovat knipl
- Problémy na pomezí slabého designu a SW chyb
  1. **Signalizace přechodu mezi režimy FBW**
     a) V Normal Law (I) nedovolí přetažení
     b) V Alternate Law (II) dovolí přetažení
     c) Přechod z (I) do (II) je signalizován jen 1 pípnutím, malým křížkem na LCD, malým nápisem
  2. **Potlačení funkce signalizace „STALL“**
     a) Při nízké rychlosti SW úmyslně potlačí alarm „STALL“
     b) Tj. při přistání... Nebo extrémním přetažení letounu!
     c) Pokud se pilot pokusí o recovery, rychlost vzroste a znovu se aktivuje alarm „STALL“ → **negativní feedback**
- **FMEA subsystému „Stall alarm“, FTA pro selhání „Stall alarm“** (false negative)

https://abcnews.go.com/International/air-france-flight-447-investigation-pilots-properly-trained/story?id=16503005

Auta s CAN sběrnicí, ovládnutí kritických funkcí hackery, 2015

- „Páteří“ moderních aut je sběrnice CAN
- Příkazy vyslanými po CAN je možné např. deaktivovat brzdy skrze signály pro ABS
- Někteří asistenti točící volantem poslouchají CAN
- Každé „online“ zařízení připojené na CAN je potenciální attack vector
- Jeep Cherokee MY14: Infotainment měl bezpečnostní zranitelnost a byl připojený na CAN
  - Úplná detaktivace funkce brzd při nízké rychlosti
  - Přepnutí vozidla na dálnici do „asistenta parkování“
- Pojišťovna požadovala zapojení telemetrického modulu do OBD-2 portu kvůli ručení, zranitelnost
  - Chevy Corvette MY14, Ford Escape MY13, Toyota Prius – ovládnutí stěračů, deaktivace funkce brzd
- **FTA, FMEA, Sanitizace vstupů, SecTest**



Pan_Myval, used under fair-use

Wired: https://youtu.be/MK0SrxBC1xs
IHS Markit – Security and the connected car
https://www.autickar.cz/blog/clanek/odborna-prednaska-hackovani-a-ovladnuti-modernich-aut/529/

Jak dělat software QA?

Prerekvizity

1) **Možnost podrobit produkt testům**
   - Přístup k existujícímu System Under Test v otestovatelném stavu
   - Všechna oprávnění potřebná k testování
   - Vhodná testovací data
2) **Nástroje**
   - pro testování, správu výsledků, a automatizaci
   - pro identifikaci a/nebo pokrytí rizik
3) **Jasná kritéria co je správně a co ne (¬ambiguity)**
   - funkční specifikace
   - nezávislé korektní výsledky pro křížové ověření
4) **Lidi s dovednostmi**
   - Test Analytici, Test Automatoři, Revieweři
   - U větších projektů QA a Risk manageři, Test Architekti
   - Testeři pozorně a svědomitě provádějící testy – nebo Test Automatizace (a tým Taut+Tsustained)
5) **Čas**

ca technologies

Obecný proces QA

- **Rešerše**
  - Co a jak má systém dělat
  - Jaké mají/mohou být vstupy, výstupy
- **Test analytika**
  - Jaké scénáře mohou nastat
  - Co musí SUT zvládnout, jaké chyby nesmí mít
  - Odstranění všech zjištěných nejasností
- **Test design + Implementace**
  - Jednotlivé Test Casy = nastavení, vstupy a očekávané výstupy
- **Spouštění**
  - Reálně provést jednotlivé Test Casy – získat a nastavit systém, ověřit funkci, vykonat testy
- **Vyhodnocení**
  - Správnost výsledků, false positives, false negatives

ca technologies

Odstranění nejednoznačností

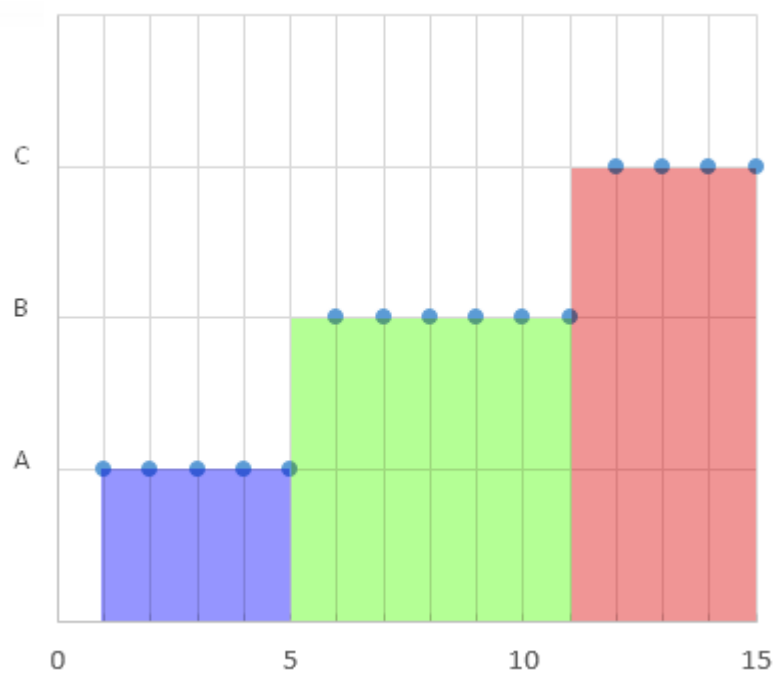
- Pokud není určen správný výstup/chování, je nemožné testovat (vyhodnotit testy)
- Pro návrh a vývoj systému je potřeba mnoho
  - drobných rozhodnutí o správné reakci systému
  - vymezení okrajových podmínek funkce
- Vývojáři je většinou automaticky řeší *svými osobními* předpoklady
- Někdy se jedná o podvědomé předpoklady („nikdy nevypadne proud“)
- verifikace napsaného kódu automaticky považuje tyto předpoklady za správné
- QA musí identifikovat všechny tyto předpoklady a *validovat* je (ideálně před QC fází)
- Standardní příklad: „systém bude přijímat a odesílat e-maily“

Bug reporting

- QA je pomocník, nikoli divadelní kritik
- Pokud chování SUT neodpovídá předpokladům (Bug/Defect/Test Anomaly...):
  1. QA ověří předpoklady (Expected results), chybu ve svých testech
  2. QA ověří zda se jedná o systematickou nebo nahodilou chybu
     a) V případě systematické chyby zdokumentuje Recreate
     b) V případě nahodilé se pokusí co nejvíce zachytit snapshot stavu systému kdy se chyba projevila
  3. Zdokumentuje relevantní verze, proměnné, systémové proměnné a nastavení, prostředí...
  4. Zdokumentuje v čem spočívá amomálie chování SUT
  5. Předá bug report vývojáři
  6. Pokud Dev udělá opravu, QA ověří že recreate už je OK, a provede regresní testy

Třídy Ekvivalence a Boundary Values

- Všechny hodnoty vstupní proměnné, se kterými program zachází stejně
- Pro základní testování je potřeba každou třídu ekvivalence otestovat právě jednou
- Krom toho je nutné otestovat krajní meze kvůli chybám v logice (cykly, záměna > a >= atd.)
- Pokud by byly otestovány všechny vstupy se všemi jejich EP, měl by test 100% path coverage
- Jenže to znamená $vstup1^{EPvstup1}$ x $vstup2^{EPvstup2}$ x ... x $vstupn^{EPvstupn}$
- To se řeší technikami test analýzy pro redukci TC (Dependency Islands apod.)




```
IF(X>0 && X<=5)
        A();
ELSE IF(X>5 && X<11)
        B();
ELSE IF(X>=11)
        C();
```

Úrovně vyspělosti QC/QA

0: Debugging	1: Verifikace	2: Hledání chyb	3: Snížení rizik	4: QA Prevence
Nespadlo to? Hotovo!	Prokázat že SUT splňuje specifikace	Level 1 a navíc:	Levely 1+2 a navíc:	Levely 1+2+3 a navíc:
(Pár náhodných vstupních dat?)	Prokázat že SUT s use cases zákazníka	Boundary Values Testing	Procesy pro pokrytí (Test Classes)	Expertní modely chyb (FMEA/FTA...)
	Prokázat že SUT funguje s běžnými stukturami/vstupy	Human Operator Error Testing	Equivalence Classes Dependency Islands	Zpětná vazba defektů = RCA SW+QA
	Client and data compatibility testing	Failure Mode Testing	Integration Testing + Environmental Testing	State Transition Testing
		Stress/System Capacity Testing	Fault Insertion Testing	Data Flow Testing
				Integrita→Fail-safe

ca technologies

Rozšířené důvody neuceleného testování

- **Bugy najdou zákazníci, my je jen musíme rychle opravit**
  - „*enterprises will be able to receive feature updates after their quality and application compatibility has been assessed in the consumer market*" –Microsoft
  - „ *error, if addressed within 30 minutes of occurring, will be quickly forgotten*" –Devskiller
- **Improvizací a experimenty vpřed**
  - „Move fast and break things" –Zuckerberg
- **Unit testy nahradí robustní testování**
  - „*robust approach to testing, which is costly (...) TDD ensures that right from the start, your codebase includes robust unit-test coverage. This guarantees that the system will work as the developer intended*" –Devskiller
- **Automatizované testy místo QA procesů a analytiky**
  -
```
given  coconuts are ripe and we have coconut tree
when   you shake coconut tree
then   coconuts fall
```

ca technologies

Q & A

Odkazy a zdroje pro inspiraci

- Hidden secrets of software QA: youtu.be/5yb55fd0nME
- Dotazy: qa@tulon.cz
- NASA Software Safety Guidebook (NASA-GB-8719.13)
- Joint SW Systems Safety Engineering Handbook
- IEEE 29119 (IEEE 829-2008, BS-7925-2)