Programy ohrožující bezpečnost

v této části se budeme zabývat pouze aplikačními programy
programy mohou poškodit či zcela zničit data, způsobit kompromitaci utajovaných skutečností, omezit a popřípadě vyloučit funkčnost celého systému.

Útoky proti integritě a utajení dat

Trapdoors

pojmem označujeme nedokumentovaný vstup do programového modulu, nejčastěji bývá vytvořen v rámci tvorby tohoto modulu za účelem snazšího ladění:

- doplnění příkazu do množiny příkazů, které modul normálně vykonává. Doplněný příkaz např. aktivuje ladicí tisky apod.
- špatně ošetřené vstupy modulu, modul nerozezná nepřípustná vstupní data, ale nechová se a nich korektně
- někdy jsou trapdoors nutné pro provedení auditu systému

trapdoors jsou obvykle odstraněny před dokončením modulu, ale mohou být opomenuty, ponechány za účelem ladění dalších modulů či snazší správy dokončeného programu, nebo pro získání neoprávněného přístupu k běžícímu programu

Trojské koně (trojan horses)

program, který kromě svých "řádných" funkcí vykonává ještě další skryté akce

objevit Trojského koně v programu o stovkách tisíc či milionech řádků je velmi obtížné, navíc v tomto smyslu může být program upraven až následně po otestování a zařazení do provozu

Salámový útok (salami attack)

jde o programy, které se snaží využívat ve svůj prospěch malých zaokrouhlovacích chyb na hranici přesnosti počítače

…ve velkém množství

program, převádějící na programátorovo konto zaokrouhlovací chyby při výpočtu úroků

- program zajišťující bezhotovostní platby, který čas od času zvýší klientovi poplatky o malou sumu, kterou pak převede na vhodný účet

- upravený scheduler, spouštějící vybrané programy v čase běhu jiných

salámový útok je opět velmi těžko detekovatelný, neboť bývá prováděn v rozsáhlých SW systémech, navíc nepůsobí viditelné problémy

objevení často pouze náhodné

Skryté kanály (covert channels)

v prostředích spravujících klasifikované informace zpravidla aplikační programátoři po ukončení vývoje nemají přístup k běžícím programům
chtějí-li získat přístup ke zpravovaným informacím, vytvoří skrytý kanál
implementace je naprosto obecná:

- zdánlivé chyby na výpisech
- existence či neexistence specifických souborů
- vznik jistých systémových událostí
- nepatrné změny front-endu
- …

zvláště vhodné pro únik malého množství informace, opět prakticky nedetekovatelné

(Objevené) slabiny programů (exploits)

nekorektní zpracování vstupních dat může vést k pádu systému, případně k totální ztrátě kontroly nad systémem
existuje obrovské množství automatických nástrojů, které i laikovi umožňují využívat známe slabiny obvyklých programů

Útoky proti dosažitelnosti služeb systému

Hladové programy (greedy programs), DOS útoky

- na mnoha počítačích běží s velmi nízkou prioritou programy, které vykonávají nejrůznější zdlouhavé výpočty, které "nepospíchají"
  nechtěné či úmyslné zvýšení priority může znamenat zahlcení celého systému
- dalším případem procesy generující velké množství synovských procesů, mnohdy chybou programu

- programy běžící v nekonečné smyčce
  operační systémy často obsahují obranné mechanimy, které násilně ukončí programy, jež běží příliš dlouho
- v době provádění I/O operace neběží virtuální čas procesu, tedy procesy generující velmi velké množství I/O operací mohou běžet takřka neomezeně dlouho
- různé druhy síťových operací a aktivit

Viry

(relativně malý) program s autoreprodukční schopností

- zpravidla se připojí nebo nahradí část kódu napadeného programu, při spuštění tohoto programu se nejprve provede kód viru, který se nainstaluje do paměti a převezme nebo pozmění některé funkce systému
- viry často obsahují obranné mechanismy proti detekci, jsou schopny instanci od instance podstatným způsobem měnit vlastní kód, po nainstalování do paměti provedou operace, jež ostatním programům učiní použitou oblast paměti nedostupnou
- velmi často po určitou dobu pouze provádějí reprodukci bez jakýchkoliv vedlejších projevů - v době odhalení tak může být napadena drtivá většina programů
- šíření virů částečně omezuje nástup systémů poskytujících ochranu paměti a dokonalejší správu prostředků, tyto systémy však bývají komplikované a dokáží tak poskytnout úkryt mnohem komplexnějším virům
- následky virové nákazy mají nejrůznější podobu - od převážně neškodných zvukových či obrazových efektů, které pouze rozptylují obsluhu a zatěžují počítač až po rozsáhlá poškození či zničení veškerých dat a programů
- podmínkou šířená virů je neopatrná manipulace s programovým vybavením, přenášení většinou nelegálního software atp.
- dobrou obranou je kromě proškolení personálu též rozdělení veškerých programů a souvisejících dat do oddílů, které jsou navzájem dostatečně odděleny, tak aby nemohlo docházet k šíření virů z jednoho oddílu do druhého.

Červy (worms)

síťová obdoba virů, mají schopnost prostřednictvím komunikačních linek se šířit z jednoho počítače na druhý

- obecně vzato mají stejné zhoubné účinky jako viry, avšak díky schopnostem samovolně se šířit v dnes již celosvětovém měřítku je jejich expanze daleko rychlejší (řádově hodiny!) a dopad tedy daleko větší
- obranou je rovněž kvalitní správa programového vybavení, používáni pouze dobře otestovaných programů a rozdělení sítě na domény, mezi kterými dochází k minimálnímu sdílení informací, které je navíc podrobeno důkladné kontrole

Metody vývoje bezpečného programového vybavení

programátoři mají k dispozici mnoho prostředků jak překonat obranné mechanismy systému
při vývoji software je tedy nutné používat metody eliminující tyto snahy

Modularita, zapouzdření, ukrytí informací

- program má být rozdělen na malé navzájem nezávislé moduly
  $\rightarrow$ výhodou snadná udržovatelnost, srozumitelnost, znovupoužitelnost, opravitelnost, testovatelnost
- moduly mezi sebou mají minimum vazeb, veškeré interakce se odehrávají přes precizně definované a *zdokumentované* rozhraní
- stačí, aby všechny moduly měli definovaný vstup, výstup a funkci, je nežádoucí šířit způsob, *jak* modul svoji funkci provádí - nemůže tak docházet k úmyslným pozměňováním ostatních modulů

Nezávislé testování

je obtížné prokazovat správnost programu - to že nebyly nalezeny chyby může pouze znamenat, že byla použita nevhodná metodika testování
testování by měl provádět nezávislý tým - snižuje se nebezpečí, že výsledný program obsahuje nežádoucí kód, či že nebyl dodržen původní cíl projektu.

Správa verzí a konfigurací (Configuration management)

hlavním cílem je zajištění dostupnosti a používání správných verzí software
občas je potřeba vrátit se k verzi před provedením jistých úprav, tento problém je závažnější v prostředí, kde na projektu pracuje celý tým
$\Rightarrow$Správa konfigurací zajišťuje udržování integrity programů a dokumentace, veškeré změny jsou vyhodnocovány a zaznamenávány

⇒Správa konfigurací zabraňuje úmyslným změnám již odzkoušených programů (vkládání trapdoors, logických bomb, …)

Hlavní důvody pro zavedení správy konfigurací

1. Zabraňuje nechtěným ztrátám předchozích verzí software
2. Odstraňuje komplikace při vývoji několika podobných verzí (např. pro různé platformy) zároveň
3. Poskytuje mechanismus pro kontrolované sdílení modulů, z nichž je skládán vytvářený systém

Za účelem udržení přehledu je nutné vést důkladnou dokumentaci všech kopií, správou konfigurací se zpravidla zabývá vyčleněný pracovník.

Programátor pracuje na vlastním exempláři modulu, který po ukončení etapy jej předá správě konfigurací včetně popisu provedených úprav a celkové charakteristiky a dále již v něm nemůže činit změny.

Je vhodné, aby správce konfigurací přijímal programy výhradně ve zdrojové formě se soupisem a popisem provedených změn. Nutné vést detailní log co, kdo, kdy dělal.

Spolehlivý software

Program je *funkčně korektní* pokud vykonává správně všechny očekávané funkce a nic víc.
Spolehlivým software (trusted software) rozumíme programy o kterých věříme, že jsou funkčně korektní a že tuto korektnost vynucují i modulů, které samy spouštějí.

operační systém je zpravidla spolehlivý sw.

Vlastnosti spolehlivých programů

- Funkční korektnost - viz. výše
- Zajištění integrity - pgm. zachová korektnost používaných dat i v případě nesprávných příkazů nebo příkazů zadaných neoprávněnými uživateli
- Omezená práva - program mající přístup k utajovaným datům minimalizuje přístup k těmto datům, přístup nepostupuje dalším ne-spolehlivým programům
- Vhodná úroveň oprávnění - program byl zkoušen v souladu s mírou utajení dat, která spravuje a s ohledem na prostředí, ve kterém běží

Spolehlivý sw. zajišťuje přístup k citlivým datům pro (obecně nespolehlivé) uživatele, kterým není možné dát přímý přístup k prvotní reprezentaci dat. Dále zajišťuje provádění sensitivních operací.

Vzájemné podezřívání (Mutual suspicion)

ne všechny programy jsou spolehlivé, ani s pomocí OS není vždy možné spolehlivost vynutit
programy vytvořené podle konceptu vzájemného podezřívání pracují jako kdyby ostatní moduly byly vadné

- nevěří volajícím, že předávají korektní vstupy
- ověřují, že volaná rutina předala správná data
- s ostatními komunikují pouze prostřednictvím dobře chráněného rozhraní

Omezení (Confinement)

používají operační systémy proti podezřelým programům
podezřelý program má přísně vymezeno, jaké systémové zdroje smí používat (sandbox)
např. runas v MS Windows, chroot v unixových systémech

Parcelizace informací (Information compartement)

veškerá data a programy v systému jsou rozdělena do několika oblastí, každá informace leží právě v jedné oblasti, každý program může pracovat s daty z nejvýše jedné oblasti, do které sám patří

Access Log

systém musí zaznamenávat co, kdo, kdy, s čím a jak dlouho dělal
podle stupně utajení se volí množina zaznamenávaných aktivit, zejména je nutné zaznamenávat chyby (nepovolené přístupy, špatná hesla, …)

Administrativní nástroje ochrany

někdy může být vhodné ovlivňovat přímo lidský faktor, a ne až výsledný produkt

Standardy vývoje programů

není vhodné povolovat programátorům, aby pracovali zcela dle svého, je třeba mít na paměti, že výsledný kód musí být verifikovatelný, udržovatelný apod.

nejčastější administrativní kontroly vývoje software

Standardní návrh - obsahuje popis povolených vývojových prostředků, jazyků, metodologií
Standardy pro tvorbu dokumentace, stylu kódování, jazyka - popis, jak má výsledný kód vypadat, volby názvů proměnných, styl komentářů, …
Standardy programování - závazné popisy, jakým způsobem se provádí programátorská práce v globálním měřítku, rozpisy peer reviews, auditů programů apod.
Standardy testování - soupis používaných verifikačních metod, archivování výsledků testů …
Standardy konfiguračního managementu - způsoby výměny produktů, způsob zaznamenávání změn, …

standardy jsou důležité při týmové práci, zabraňují vzniku modulů, kterým rozumí pouze autor

Dodržování standardů vývoje programů

aby byly efektivní, musí být standardy důsledně dodržovány za všech okolností
typickými situacemi, kdy vznikají tendence je porušovat jsou okamžiky, kdy projekt nabírá zpoždění proti plánu, po odchodu klíčových pracovníků apod.
dodržování standardů by měli podporovat pravidelné audity bezpečnosti (security audits) prováděné nezávislým bezpečnostním týmem

Rozdělení úkolů

je vhodné práci rozdělit mezi více lidí, kteří se znají co možná nejméně
omezuje se tak nebezpečí vzniku bezpečnost ohrožujících programů, navíc pokud programátor očekává, že jeho kód bude podroben zkoumání nezávislého testovacího týmu, omezí své nekalé aktivity

Charakter přijímaných pracovníků

firmy běžně sestavují profily svých potenciálních pracovníků a přijímané podrobují všestrannému zkoumání, zda nebudou představovat hrozbu pro bezpečnost - obvyklé jsou reference z dřívějších působišť, psychologické testy apod.
po přijetí má pracovník povětšinou velmi omezený přístup k senzitivním informacím, až časem získává důvěru a tím i rozsáhlejší bezpečnostní oprávnění

Sledování pracovníků

je vhodné vést co nejpodrobnější informace o pracovnících, zejména o:

- vybraných zálibách (hraní, drogy, sex, jiné finančně náročné koníčky)
- neobvyklých změnách chování pracovníka
- nenadálých změnách majetkových poměrů

Verifikace a validace software

ověřuje, zda požadavky na SW jsou korektně implementovány a odpovídají systémové specifikaci, cílem je důkladně analyzovat a otestovat
v&v se provádí v průběhu a po dokončení díla s souladu s připraveným plánem, který by měl připravit tým tvořící systémovou specifikaci
provádění v&v má být prováděno nezávisle na tvorbě SW po stránce:

- technické – jiní lidé, kt. sami pochopí specifikaci, ověří užité vývojové prostředky a nejsou zatíženi znalostí průběhu návrhu řešení
- řídící – tým si musí sám zvolit co a jakým způsobem bude testovat, pouze odpovídá za sprvánost výsledků (jejichž obsah je dán v rámci plánu)
- finanční – tým musí být závislý pouze na včasnosti a správnosti dodání svých výsledků, nikoliv na výsledcích celého projektu

Řízení v&v

Pareto efekt: 20% chyb spotřebuje 80% nákladů na předělávky

zajišťuje plánování, koordinaci, včasnou identifikaci problémů, vyhodnocování výkonnosti a odhady dopadu změn návrhu, navrhuje vhodnou metriku pro hodnocení kvality sw, poskytuje zpětnou vazbu vývojářům
pro odstranění největších problémů slouží analýza hazardů a kritických sekcí

Aktivity v&v

- validace požadavků na sw – ověřit, zda požadavky nejsou v rozporu s platnými standardy, zda nejsou vnitřně sporné
- v&v návrhu sw – ověřit, že návrh úplně a bezchybně implementuje požadavky
- v&v kódu – že byly správně užity předepsané postupy a standardy vývoje
- testování – (t. modulů, t. integrace, t. systému, t. instalace) na jednotlivých úrovních se zkouší, zda výsledek se chová přesně dle zadání, zejména pod tlakem
- v&v při správě a používání sw – co dělat při změně v systému,

Major Software V&V Activities

Aktivita/fáze	Úkoly
Řízení V&V	Plánování
	Monitoring
	Vyhodnocení výsledků, vliv změn
	Reporting
Software Requirements V&V	Kontrola dokumentace konceptů
	Analýza trasovatelnosti
	Vyhodnocení požadavků na software
	Analýza rozhraní
	Iniciální plánování systémových testů software
	Reporting
Software Design V&V	Analýza trasovatelnosti
	Vyhodnocení návrhu software
	Analýza rozhraní
	Iniciální plánování modulových testů
	Iniciální plánování integračních testů
	Reporting
Code V&V	Analýza trasovatelnosti
	Vyhodnocení kódu
	Analýza rozhraní
	Dokončení přípravy modulových testů
	Reporting
Unit Test	Provedení modulových testů
	Reporting
Software Integration Test	Dokončení přípravy integračních testů
	Provedení integračních testů
	Reporting
Software System Test	Dokončení přípravy systémových testů
	Provedení systémových testů
	Reporting
Software Installation Test	Auditování koordinace při instalaci
	Reporting
Software Operation and maintenance V&V	Analýza vlivu změny
	Opakování řízení V&V
	Opakování technických aktivit V&V

v&v techniky

statické – přímo zkoumají struktury a formu produktu bez jeho spuštění (reviews, inspekce, data-flow)
dynamické – analýza výsledků zkušebních běhů a simulací (testování, prototypování)
formální – matematická analýza zadání a funkčnosti (VDM, Z)

Analýza algoritmů *(Algorithm analysis)* – zpětným přepisem do běžného jazyka nebo formalismu je ověřována logika a správnost návrhu, zahrnuje znovuodvození vztahů, vhodnost návrhu, stabilitu, časování, přetypování atd. *Cíle: přesnost; efektivita; správnost;časování; prostorová náročnost.*

Analytické modelování *(Analytic modelling)* – vyhodnocení výkonu a plánování zdrojů. *Cíle: přesnost; efektivita; úzká místa; proveditelnost; predikce výkonu.*

Back-to-back testing porovnávání výstupu více programů se stejnou specifikací. *Cíle: anomality, rozdíly mezi verzemi.*

Analýza mezních hodnot *(Boundary value analysis)* –detekování chyb na mezních vstupech. (nuly, prázdná pole a řetězce, první a poslední položka…). *Cíle: analýza algoritmů; velikost polí; inkonzistence limitů; chyby specifikace.*

Čtení kódu *(Code reading)* – expertní pročítání cizího kódu. *Cíle: korektnost; užívání proměnných; nevolané funkce; testy parametrů; styl; redundance.*

Analýza toku řízení *(Control flow analysis)* na grafické reprezentaci se studuje předávání řízení běhu, hierarchie volání rutin, dosažitelnost stavů. *Cíle: úzká místa; testy hranic; struktura modulů; korektnost; vyhodnocení návrhu; souborové operace;vyhodnocení formální specifikace; tok informací a konzistence; interakční testy; invarinty cyklů; efektivita; predikce výkonu.*

Analýza pokrytí *(Coverage analysis)* určí míru otestování systému danou sadou testů. *Cíle: integrační testy, systémové testy.*

Kritická analýza *(Critical timing/flow analysis)* ověřuje dodržení časových nároků v návrhu. *Cíle: modelování; synchronizace; časování.*

Databázová analýza *(Database analysis)* ověřuje, že DB schéma a procedury odpovídají logickému návrhu. Integrita dat. *Cíle: ochrana přístupu; typování; vyhodnocování návrhu; souborové operace; tok informací; efektivita; prostorové nároky;testy modulů.*

Analýza toku dat *(Data flow analysis*) při návrhu globální struktury aplikace. Čtení proměnných až po inicializaci, opakované zápisy bez čtení, … *Cíle: úzká místa; testy mezí; hierarchie modulů; reakce na okolí; šíření chyb; souborové operace; tok informací; invarinty cyklů; efektivita; vyhodnocování návrhu; integrační testy; predikce výkonu; neinicializované hodnoty; odkazy.*

Rozhodovací tabulky *(Decision (truth) tables*) analýza komplexních logických vztahů a rozhodování. *Cíle: logické chyby.*

Desk checking – hledání zjevných chyb v kódu. *Cíle: volání neexistujících funkcí; meze polí; neshoda s návrhem; práce s registry; špatné linkování; nekonečné cykly; vadná inicializace; obrácené predikáty; neshoda parametrů; nekonečná rekurze; nedeklarované proměnné; nedosažitelný kód.*

Error seeding podsouváním známých chyb do programu zjišťuje, zda navržené testy jsou adekvátní

$$\frac{\text{Nalezených vsunutých chyb}}{\text{Celkem vsunutých chyb}} = \frac{\text{Nalezených skutečných chyb}}{\text{Celkem skutečných chyb}}$$

Cíle: adekvátnost testů.

Event tree analysis analýza zdola nahoru, sledování jak se rozšiřuje vliv události na systém. *Cíle: analýza hazardů; bezpečnost; analýza hrozeb; časování.*

Konečné automaty *(Finite state machines* FSM) modeluje system konečným automatem, popisuje reakce na vstupy *Cíle: úplnost specifikace; nekonzistentní požadavky; modelování.*

Funkční testování *(Functional testing*) spuštěním systému se zkoumá, zda jsou naplněny požadavky uživatelů. *Cíle: testy mezí; souborové operace; provozní charakteristika; pokrytí specifikace; predikce výkonu; systémové testy; test modulů; neinicializované hodnoty; odkazy; trasování proměnných.*

Inspekce *(Inspections)* skupina expertů nezávisle analyzuje kód, následně uspořádají monitorované sezení, kde jsou identifikovány a zaznamenány nalezené nedostatky. Následně se provádí kontrola jejich odstranění. *Cíle: přesnost; naplnění požadavků; vyhodnocení formální specifikace; konzistence toku informací; logické chyby; invarinty cyklů; prostorové nároky; syntaktické chyby; neinicializované hodnoty.*

Analýza rozhraní *(Interface analysis*) statické ověření, že rozhraní modulů jsou správná, reagují na všechny možné vstupy, jsou správně volány, dodržují dané normy atd. *Cíle: nesoulad parametrů; použití globálních proměnných; nesprávné užití statických a dynamických dat; volání nesprávných funkcí; chyby popisu I/O.*

Testování rozhraní *(Interface testing*) dynamická obdoba předchozí metody. *Cíle: nesoulad parametrů; použití globálních proměnných; nesprávné užití statických a dynamických dat; volání nesprávných funkcí; chyby popisu I/O.*

Analýza mutací *(Mutation analysis*) ověřuje vhodnost testů. Vnesením záměrných chyb je vytvořena řada mutací původního systému a na každou z nich se aplikují testy. Srovnávají se výsledky. *Cíle: testy mezí; retestování po změně; příprava testů.*

Testování výkonu *(Performance testing)* porovnává skutečný výkon s pořadavky, zatížení komponent, … *Cíle: alokace; synchronizace; časování.*

Petriho sítě *(Petri-nets*) ověřuje odolnost návrhu proti deadlockům, proveditelnost, dosažitelnost. Systém je modelován za použití stavů, událostí, přechodů, vstupů a výstupů. Je možné simulovat běh systému za různých podmínek. *Cíle: analýza hazardů; modeling; bezpečnost; analýza hrozeb; časování.*

Důkaz korektnosti *(Proof of correctness* - formální verifikace) matematickými metodami se modeluje systém a formálně se dokáže splnění nakladených předpokladů a tvrzení o funkčnosti. *Cíle: korektnost; ověření kritických sekcí.*

Prototypování *(Prototyping*) zkoumání pravděpodobných výsledků implementace - identifikace nekompletních a nepsrávných požadavků, ověřování vhodnosti návrhu. *Cíle: chování; nepokryté požadavky na funkcionalitu; úplnost specifikace; uživ. rozhraní.*

Regresní analýza a testy *(Regression analysis and testing*) znovuvyhodnocení naplnění (původních) požadavků po větších změnách. *Cíle: integrační testy; retestování po změně; systémové testy; testy modulů.*

Procházení požadavků *(Requirements parsing)* zkoumání, zda všechny požadavky jsou specifikovány jednoznačně a úplně *Cíle: přesnost; checklisty; kompletnost; 8zkonzistence; proveditelnost; vyhodnocení formální specifikace; konzistence toku informací; integrační testy; korektnost; vyhodnocení naplnění požadavků; retestování po změně; pokrytí specifikace; systémové testy; testy modulů.*

Reviews kontroly naplnění požadavků, návrhu software, často jako předpoklad provedení dané vývojové aktivity *Cíle: před valstními testy, logické chyby; syntaktické chyby.*

Sensitivity analysis je predikcí pravděpodobnosti, s jakou testování odhalí chyby. Umožňuje porovnávat různé testovací strategie a odhadovat, které oblasti kódu budou dotknuty změnou v rámci údržby. Lze použít na hodnocení odolnosti vůči chybám. *Cíle: korektnost; logické chyby; reliability; adekvátnost testů.*

Simulation – využívá se na zkoumání interakcí velkých komplexnízh systémů. Využívá spustitelný model a zkoumá chování systému a reakci na práci administrátora. *Cíle: chování; testy mezí; testování verzí; sampling, support; proveditelnost; souborové operace; path testing; provozní charakteristika; retestování po změně; pokrytí specifikace; predikce výkonu; systémové testy; neinicializované hodnoty; odkazy; kontroly/trasování proměnných.*

Sizing a analýza časování – určování, zda alokace SW a HW jsou dostatečné pro daný návrh. *Cíle: efektivita algoritmů; úzká místa; testy mezí; testování verzí; integrační testy; efektivita; provozní charakteristika; retestování po změně; prostorové nároky; systémové testy; časování;testy modulů.*

Slicing je technika dekompozice programů, kdy výstupní hodnota je trasována zpětně programem za účelem identifikace všech relevantních součástí kódu. Užitečné pro demonstraci funkční diveristy. *Cíle: alokace V&V zdrojů; společný kód; konzistence toku informací; dekompozice programu; odkazy.*

Chybový mód, efekty, kritická analýza *(Software failure mode, effects and criticality analysis)* odhaluje nedostatečné naplnění požadavků zkoumáním důsledku chyby modulu (vč. vadných instrukcí) v závislosti na typu chyby.

Výsledkem je matice dopadů různých chyb na systém popisující rozsah, kritičnost, potřebné změny, preventivní opatření. *Cíle: analýza hazardů; bezpečnost; úplnost specifikace; analýza hrozeb.*

Analýza chybových stromů *(Software fault tree analysis)* určuje a analyzuje požadavky na bezpečnost. Užívá se k určení dopadů možných rizik. Cílem ověřit, že SW nepřivede systém do nebezpečného stavu a za jakých podmínek případně hrozí nebezpečí. Analytik modeluje situaci, ze hrozba byla realizována a zkoumá, jaké okolnosti ji mohly způsobit. *Cíle: analýza hazardů; bezpečnost; analýza hrozeb.*

Stress testing zkoumá odezvu systému až do extrémních podmínek, aby se dohalily slabiny a demonstrovala shopnost zvládat normální zátěž. *Cíle: chyby návrhu; plánování nastavení prosystém při přetížení.*

Structurální testování posuzuje logiku modulů, může být použito pro ověření naplnění požadavků pro testy pokrytí, tj. jak moc byl program „prohnán". *Cíle: úzká místa; šíření chyb; kontrola parametrů; provozní charakteristika; retestování po změně.*

Symbolické spouštění *(symbolic execution)* ověřuje shodu mezi kódem a specifikací. Spuštění programu je simulováno nad symboly namísto dat, výstup je vyjádřen jako výraz nad těmito symboly. *Cíle: testování tvrzení; provozní charakteristika; důkaz korektnosti; retestování po změně.*

Certifikace testů zajišťuje, že výsledky testů odpovídají skutečným nálezům. Použité nástroje, media a dokumentace jsou certifikovány pro zajištění udržovatelnosti a opakovatelnosti. Ověření, že dodaný produkt je identický s objektem V&V. *Cíle: dodávka nesprávné verze produktu; nesprávné výsledky; vynechané zprávy a testy.*

Procházky *(Walkthroughs)* podobné reviews, méně formální, více detailní. Autor vede skupinu expertů návrhem, nebo kódem modulu, ostatní komentují použitou techniku, styl, hledají chyby, porušení standardů, ptají se na detaily. *Cíle: checklisty; šíření chyb; vyhodnocení formální specifikace; go-no-go rozhodnutí; logické chyby; manuálí simulace; kontrola parametrů; retestování po změně; malé ale složité nebo k chybám náchylné sekce; kontroly stavu; syntaktické chyby; systémové testy; technické kontroly.*

V&V pro znovupoužitý SW

Většina V&V technik je vhodná pro znovupužitý SW, následující specifické techniky je vhodně doplňují

Analýza konzistence porovnává požadavky na veškerý stávající SW s novou specifikací pro zajištění konzistence. *Cíle: konzistence.*

Analýza rozhraní *(Interface analysis)* (viz analýza rozhraní a testování rozhraní výše) mimořádně důležité pro zkoušky rozhraní znovupoužitých modulů pro zajištní korektnosti adaptace a zjištění případných rozdílů.

Techniky specifické pro báze znalostí (KBS)

Alternativní model srovnává doménový model implementovaný daným KBS s alternativním modelem na úplnost a přesnost.
Control groups mohou být použity v rámci testování pro porovnáví výkonu při zpracování úlohy s/bez KBS
Analýza kredibility porovnává výsledky systému s odpověďmi experta.
Field testing – KBS se reálně používá, zaznamenávají s vyhodnocují se výsledky.
Testování nepovolených atributů - kontrola pravidel na omezení nepovolených hodnot atributů.
Logická verifikace jde o ověření kompletnosti expertní znalosti a konzistence v době vytváření doménového modelu.
Meta modely - porovnávání znalostí a pravidel s metamodelem domény.
Partition testing - vybírá příklady pro různé oblasti vstupních a výstupních hodnot a ověřuje, že specifikace pokrývá tyto případy
Verifikace pravidel zkoumá se kompletnost, vnořená/reduntantní pravidla, nekonzistentní pravidla, pravidla, která nemohou být splněna, cyklická pravidla, nedosažitelné výsledky, apod.
Statistická validace zkoumá, jak často používá pravidla nebo skupiny pravidel v bázi znalostí.
Turingův test - naslepo porovnává výkon systému s výkonem experta.
Weight analysis porovnává statistické informace spojené s pravidlem se statickou znalostí o dané doméně.