# Certifikovaný tester základnej úrovne Učebná osnova

*v4.0.1*

---

International Software Testing Qualifications Board

---


---

Poskytol Czech and Slovak Quality Board

CaSQB Czech and Slovak Quality Board

# Upozornenie o ochrane autorských práv

# História zmien

| Slovenská verzia | Dátum | Poznámka |
|---|---|---|
| CTFL SK v4.0.1 | 15. 8. 2024 | Preklad QC v kap 0.12 zmenený z "kontrola kvality" na "riadenie kvality". Zmena názvu kapitoly 1.1.1. |
| CTFL SK v4.0 | 1. 7. 2024 | Certifikovaný tester základnej úrovne – slovenský preklad. |
| CTFL SK v4.0 Beta | 15. 3. 2024 | Certifikovaný tester základnej úrovne – slovenský preklad – Beta verzia. |
| CTFL SK 2018 v3.1 | 6. 4. 2020 | Nová verzia podľa ISTQB® CTFL v3.1. Opravy preklepov a formátovania. |
| CTFL SK 2018 v3.1 Beta 1 | 19. 2. 2020 | Aktualizácia verzie 2018 podľa ISTQB® CTFL verzie 3.1. |
| CTFL SK 2018 v2.1 Beta 1 | 3. 2. 2020 | Nový slovenský preklad verzie 2018 s grafickými úpravami. |
| CTFL SK 2018 v1.2 Alfa 2 | 30. 1. 2020 | Nový slovenský preklad verzie 2018 s jazykovými korekciami. |
| CTFL SK 2018 v1.1 Alfa 1 | 12. 1. 2020 | Nový slovenský preklad verzie 2018. |
| CTFL SK 2011 v0.3 Beta 3 | 28. 2. 2019 | Certifikovaný tester základnej úrovne – slovenský preklad – Beta 3. |
| CTFL SK 2011 v0.2 Beta 2 | 9. 10. 2011 | Certifikovaný tester základnej úrovne – slovenský preklad – Beta 2. |
| CTFL SK 2011 v0.1 Beta 1 | 30. 9 . 2011 | Certifikovaný tester základnej úrovne – slovenský preklad – Beta 1. |
| CTFL SK 2007 Beta 2 | 10. 3. 2008 | Certifikovaný tester základnej úrovne – slovenský preklad – Beta 2. |
| CTFL SK 2007 Beta 1 | 1. 2. 2008 | Certifikovaný tester základnej úrovne – slovenský preklad – Beta 1. |

| Anglická verzia | Dátum | Poznámka |
|---|---|---|
| CTFL v4.0 | 21. 4. 2023 | CTFL v4.0 – General release version |
| CTFL v3.1.1 | 1. 7. 2021 | CTFL v3.1.1 – Copyright and logo update |
| CTFL v3.1 | 11. 11. 2019 | CTFL v3.1 – Maintenance release with minor updates |
| ISTQB® 2018 | 27. 4. 2018 | Candidate general release version |
| ISTQB® 2011 | 1. 4. 2011 | Certified Tester Foundation Level Syllabus Maintenance Release – see Release Notes |
| ISTQB® 2010 | 30. 3. 2010 | Certified Tester Foundation Level Syllabus Maintenance Release – see Release Notes |
| ISTQB® 2007 | 1. 5. 2007 | Certified Tester Foundation Level Syllabus Maintenance Release |
| ISTQB® 2005 | 1. 7. 2005 | Certified Tester Foundation Level Syllabus |
| ASQF V2.2 | July 2003 | ASQF Syllabus Foundation Level Version 2.2 "Lehrplan Grundlagen des Software-testens" |
| ISEB V2.0 | 25. 2. 1999 | ISEB Software Testing Foundation Syllabus V2.0 |

# Obsah

# Poďakovanie ISTQB®

Tento dokument bol formálne vydaný Valným zhromaždením ISTQB® dňa 21. 4. 2023.

Bol vytvorený spoločným tímom pracovných skupín ISTQB® Foundation Level a Agile Working Groups v zložení: Laura Albert, Renzo Cerquozzi (podpredseda), Wim Decoutere, Klaudia Dussa-Zieger, Chintaka Indikadahena, Arnika Hryszko, Martin Klonk, Kenji Onishi, Michaël Pilaeten (podpredseda), Meile Posthuma, Gandhinee Rajkomar, Stuart Reid, Eric Riou du Cosquer (podpredseda), Jean-François Riverin, Adam Roman, Lucjan Stapp, Stephanie Ulrich (podpredsedníčka), Eshraka Zakaria. Tím ďakuje Stuartovi Reidovi, Patrícii McQuaidovej a Leanne Howardovej za technickú revíziu a revíznemu tímu a členským výborom za ich návrhy a príspevky.

Na revízii, pripomienkovaní a hlasovaní o týchto učebných osnovách sa podieľali: Adam Roman, Adam Scierski, Ágota Horváth, Ainsley Rood, Ale Rebon Portillo, Alessandro Collino, Alexander Alexandrov, Amanda Logue, Ana Ochoa, André Baumann, André Verschelling, Andreas Spillner, Anna Miazek, Armin Born, Arnd Pehl, Arne Becher, Attila Gyúri, Attila Kovács, Beata Karpinska, Benjamin Timmermans, Blair Mo, Carsten Weise, Chinthaka Indikadahena, Chris Van Bael, Ciaran O'Leary, Claude Zhang, Cristina Sobrero, Dandan Zheng, Dani Almog, Daniel Säther, Daniel van der Zwan, Danilo Magli, Darvay Tamás Béla, Dawn Haynes, Dena Pauletti, Dénes Medzihradszky, Doris Dötzer, Dot Graham, Edward Weller, Erhardt Wunderlich, Eric Riou Du Cosquer, Florian Fieber, Fran O'Hara, François Vaillancourt, Frans Dijkman, Gabriele Haller, Gary Mogyorodi, Georg Sehl, Géza Bujdosó, Giancarlo Tomasig, Giorgio Pisani, Gustavo Márquez Sosa, Helmut Pichler, Hongbao Zhai, Horst Pohlmann, Ignacio Trejos, Ilia Kulakov, Ine Lutterman, Ingvar Nordström, Iosif Itkin, Jamie Mitchell, Jan Giesen, Jean-Francois Riverin, Joanna Kazun, Joanne Tremblay, Joëlle Genois, Johan Klintin, John Kurowski, Jörn Münzel, Judy McKay, Jürgen Beniermann, Karol Frühauf, Katalin Balla, Kevin Kooh, Klaudia Dussa-Zieger, Klaus Erlenbach, Klaus Olsen, Krisztián Miskó, Laura Albert, Liang Ren, Lijuan Wang, Lloyd Roden, Lucjan Stapp, Mahmoud Khalaili, Marek Majernik, Maria Clara Choucair, Mark Rutz, Markus Niehammer, Martin Klonk, Márton Siska, Matthew Gregg, Matthias Hamburg, Mattijs Kemmink, Maud Schlich, May Abu-Sbeit, Meile Posthuma, Mette Bruhn-Pedersen, Michal Tal, Michel Boies, Mike Smith, Miroslav Renda, Mohsen Ekssir, Monika Stocklein Olsen, Murian Song, Nicola De Rosa, Nikita Kalyani, Nishan Portoyan, Nitzan Goldenberg, Ole Chr. Hansen, Patricia McQuaid, Patricia Osorio, Paul Weymouth, Pawel Kwasik, Peter Zimmerer, Petr Neugebauer, Piet de Roo, Radoslaw Smilgin, Ralf Bongard, Ralf Reissing, Randall Rice, Rik Marselis, Rogier Ammerlaan, Sabine Gschwandtner, Sabine Uhde, Salinda Wickramasinghe, Salvatore Reale, Sammy Kolluru, Samuel Ouko, Stephanie Ulrich, Stuart Reid, Surabhi Bellani, Szilard Szell, Tamás Gergely, Tamás Horváth, Tatiana Sergeeva, Tauhida Parveen, Thaer Mustafa, Thomas Eisbrenner, Thomas Harms, Thomas Heller, Tobias Letzkus, Tomas Rosenqvist, Werner Lieblang, Yaron Tsubery, Zhenlei Zuo a Zsolt Hargitai.

ISTQB® Working Group Foundation Level (verzia 2018): Klaus Olsen (predseda), Tauhida Parveen (podpredsedníčka), Rex Black (projektový manažér), Eshraka Zakaria, Debra Friedenberg, Ebbe Munk, Hans Schaefer, Judy McKay, Marie Walsh, Meile Posthuma, Mike Smith, Radoslaw Smilgin, Stephanie Ulrich, Steve Toms, Corne Kruger, Dani Almog, Eric Riou du Cosquer, Igal Levi, Johan Klintin, Kenji Onishi, Rashed Karim, Stevan Zivanovic, Sunny Kwon, Thomas Müller, Vipul Kocher a Yaron Tsubery. Tím ďakuje tímu revidujúcich a všetkým členským výborom za návrhy k súčasným učebným osnovám.

ISTQB® Working Group Foundation Level (verzia 2011): Thomas Müller (predseda), Debra Friedenberg. Hlavný tím ďakuje tímu revidujúcim (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquer, Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal) a

všetkým členským výborom za návrhy k súčasným učebným osnovám.

ISTQB® Working Group Foundation Level (verzia 2010): Thomas Müller (predseda), Rahul Verma, Martin Klonk a Armin Beer. Tím ďakuje tímu revidujúcim (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal) a všetkým členským výborom za ich pripomienky.

ISTQB® Working Group Foundation Level (verzia 2007): Thomas Müller (predseda), Dorothy Graham, Debra Friedenberg a Erik van Veenendaal. Tím ďakuje tímu revidujúcim (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, a Wonil Kwon) a všetkým členským výborom za ich pripomienky.

ISTQB® Working Group Foundation Level (verzia 2005): Thomas Müller (predseda), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson a Erik van Veenendaal. Tím ďakuje tímu revidujúcich a všetkým členským výborom za návrhy.

# Poďakovanie CaSQB

Preklad do slovenského jazyka, verzia v4.0: Daniel Poľan, Gabriela Spodniaková, Pavla Vaculková (preklad); Karol Frühauf, Roman Jurkech, Marek Majerník, Daniel Poľan, Juraj Žabka (alfa revízie); Marek Gejdoš, Tomáš Petráň, Dominika Sabová (beta revízie).

Preklad do slovenského jazyka, verzia 2018 v3.1: Marek Majerník, Daniel Poľan, Filip Rechtoris (preklad), Robert Dankanin, Lenka Spodniaková (revízie).

Preklad do slovenského jazyka, verzia 2018 v1.0: Marek Majerník, Daniel Poľan, Filip Rechtoris (preklad), Robert Dankanin, Karol Frühauf, Miroslav Piter, Lenka Spodniaková (revízie).

Preklad do slovenského jazyka, verzia 2011: Róbert Dankanin, Marek Majerník, Ľuboš Práznovský (preklad), Daniela Čuvarská, Marcel Veselka (revízie).

Preklad do slovenského jazyka, verzia 2007: Daniela Čuvarská, Róbert Dankanin, Karol Frühauf, Marek Majerník, Ľuboš Práznovský (preklad), Roman Jurkech (revízie).

Hoci bolo snahou prekladateľov docieliť čo najvernejší preklad pôvodného anglického vydania, priestor pre zdokonaľovanie v takom komplexnom texte určite je a stále bude. Postrehy a podnety čitateľov preto radi uvítame e-mailom na adrese *translation@castb.org*.

# 0 Úvod

## 0.1 Účel učebných osnov

Tieto učebné osnovy tvoria základ pre medzinárodnú kvalifikáciu testovania softvéru základnej úrovne. ISTQB® poskytuje tieto učebné osnovy nasledovne:

1. Členským výborom, pre preklad do daného národného jazyka a za účelom akreditácie poskytovateľov školení. Členské výbory môžu prispôsobiť osnovy konkrétnym potrebám svojho jazyka a pridať odkazy na lokálne publikácie.
2. Certifikačným autoritám, za účelom vytvorenia skúškových otázok v národnom jazyku prispôsobených študijným cieľom pre tieto osnovy.
3. Poskytovateľom školení, k príprave výukových kurzov a stanoveniu vhodných výukových metód.
4. Záujemcom o certifikáciu, za účelom prípravy na certifikačnú skúšku, a to buď ako súčasť výukového kurzu alebo na samostatnú prípravu.
5. Medzinárodnej komunite softvérových a systémových inžinierov, za účelom podpory profesie testovania softvéru a systémov, a tiež ako zdroj pre knihy a články.

## 0.2 Certifikovaný tester základnej úrovne pre testovanie softvéru

Táto úroveň je určená každému, kto sa podieľa na testovaní softvéru. Patria sem ľudia v rolách testerov, analytikov testovania, špecialistov v oblasti testovania, konzultantov, manažérov testovania, softvérových vývojárov alebo iných členov tímov. Základná úroveň je tiež vhodná pre každého, kto chce získať základné znalosti v oblasti testovania softvéru, napríklad pre vlastníkov produktov (product owners), projektových manažérov, manažérov kvality, manažérov vývoja softvéru, biznis analytikov, IT riaditeľov a konzultantov na úrovni manažmentu. Získanie certifikátu základnej úrovne oprávňuje jeho držiteľa k certifikáciám vyšších úrovní v oblasti testovania softvéru.

## 0.3 Kariérna cesta pre testerov

Program ISTQB® podporuje odborníkov v oblasti testovania na všetkých úrovniach ich kariérneho rozvoja a ponúka rozvoj znalostí ako do šírky, tak do hĺbky. Jednotlivci, ktorí získali certifikáciu ISTQB® základnej úrovne, môžu mať tiež záujem o pokročilé úrovne (analytik testovania, technický analytik testovania a manažér testovania) a následne o expertné úrovne (manažment testovania alebo zlepšovanie testovacieho procesu). Každý, kto chce rozvíjať zručnosti pre testovanie v agilnom prostredí, môže zvážiť certifikáciu Agile Technical Tester alebo Agile Test Leadership at Scale. ISTQB® tiež ponúka v rámci kariérnej cesty označovanej Specialist detailný pohľad do oblastí, v ktorých sú vyžadované špecifické prístupy k testovaniu a špecifické testovacie činnosti (napr. automatizácia testovania, testovanie AI, testovanie založené na modeloch, testovanie mobilných aplikácií) alebo ktoré súvisia s konkrétnymi testovacími oblasťami (napr. testovanie výkonu, testovanie použiteľnosti, akceptačné testovanie, bezpečnostné testovanie), prípadne kde je nevyhnutné mať špeciálne priemyselné know-how (napr. automobilový priemysel alebo hry). Aktuálne informácie o certifikáciách testerov podľa schémy ISTQB® možno nájsť na stránkach *www.castb.org* alebo *www.istqb.org*.

## 0.4 Profesijné ciele

V tejto časti je uvedených 14 profesijných cieľov, ktoré by mal spĺňať každý, kto dosiahol certifikáciu základnej úrovne. Certifikovaný tester základnej úrovne dokáže:

| Kód | Profesijný cieľ |
|---|---|
| FL-BO1 | Chápať, čo je testovanie a prečo je prospešné. |
| FL-BO2 | Chápať základné koncepty testovania softvéru. |
| FL-BO3 | Určiť prístup k testovaniu a činnosti, ktoré majú byť vykonané v závislosti od kontextu. |
| FL-BO4 | Posúdiť a zlepšiť kvalitu dokumentácie. |
| FL-BO5 | Zvýšiť efektivitu a účinnosť testovania. |
| FL-BO6 | Zladiť proces testovania so životným cyklom vývoja softvéru. |
| FL-BO7 | Rozumieť princípom manažmentu testovania. |
| FL-BO8 | Písať jasné a zrozumiteľné reporty o defekte a vedieť ich komunikovať. |
| FL-BO9 | Chápať faktory, ktoré ovplyvňujú priority a náročnosť činností súvisiacich s testovaním. |
| FL-BO10 | Pracovať ako súčasť multifunkčného tímu. |
| FL-BO11 | Rozumieť rizikám a prínosom automatizácie testov. |
| FL-BO12 | Identifikovať nevyhnutné zručnosti potrebné na testovanie. |
| FL-BO13 | Chápať vplyv rizík na testovanie. |
| FL-BO14 | Efektívne reportovať o postupe a kvalite testovania. |

## 0.5 Preskúmateľné študijné ciele a kognitívne úrovne znalostí

Študijné ciele sú odvodené od profesijných cieľov a sú použité k vytváraniu certifikačných skúšok. Vo všeobecnosti je všetok obsah kapitol 1-6 v týchto osnovách preskúmateľný na úrovni K1. To znamená, že kandidát môže byť požiadaný, aby určil, zapamätal si alebo si vybavil kľúčové slovo alebo koncept z niektorej zo šiestich kapitol týchto osnov. Úrovne znalostí konkrétnych študijných cieľov sú uvedené na začiatku každej kapitoly a sú klasifikované nasledovne:

- K1: Zapamätať si
- K2: Pochopiť
- K3: Použiť

Na začiatku každej kapitoly (tesne pod názvom kapitoly) je uvedený zoznam kľúčových slov. Všetky uvedené kľúčové slová je nutné si zapamätať (K1), a to aj v prípade, že nie sú výslovne uvedené v študijných cieľoch. Viac informácií o klasifikácii úrovní znalostí vrátane príkladov možno nájsť v *sekcii 7*.

## 0.6 Certifikačná skúška základnej úrovne

Skúška pre získanie certifikátu z modulu Certifikovaný Tester Základná Úroveň je založená na týchto učebných osnovách. Odpovede na jednotlivé otázky pri certifikačnej skúške môžu vyžadovať znalosti

z rôznych kapitol. Všetky časti týchto osnov môžu byť predmetom certifikačnej skúšky (okrem úvodu a príloh). Normy, knihy a iné osnovy ISTQB® sú uvedené ako odkazy (pozri *kapitolu 9*), ale ich obsah nie je súčasťou skúšky s výnimkou toho, čo je z nich priamo zahrnuté v týchto osnovách. Viac informácií je možné nájsť v dokumente Štruktúry a pravidlá skúšok základnej úrovne (v anglickom jazyku dostupné ako Foundation Level Examination Structures and Rules).

## 0.7 Akreditácia

Členský výbor ISTQB® môže akreditovať poskytovateľa školení, ktorého študijné materiály zodpovedajú týmto učebným osnovám. Pokyny pre akreditáciu by mali poskytovatelia školení dostať od daného členského výboru alebo orgánu, ktorý akreditáciu vykonáva. Akreditovaný kurz je následne uznaný za vyhovujúci týmto osnovám a je doň možné priamo začleniť certifikačnú skúšku ISTQB®. Pokyny pre akreditáciu týchto učebných osnov sa riadia všeobecnými akreditačnými pokynmi zverejnenými príslušnou pracovnou skupinou ISTQB® (v anglickom jazyku dostupné ako Accreditation Guidelines).

## 0.8 Využitie noriem

V týchto učebných osnovách sú uvedené odkazy na rôzne normy (napr. normy IEEE alebo ISO). Informácie v týchto normách môžu byť citované priamo (napr. norma ISO 25010 pri definícii kvalitatívnych charakteristík) alebo poskytujú prípadnému čitateľovi zdroj ďalších informácií. Samotné normy však nie sú predmetom skúšky. Viac informácií možno nájsť v *kapitolu 9*.

## 0.9 Zaistenie aktuálnosti

Softvérový priemysel sa dynamicky mení. Aby bolo možné sa s týmito zmenami vysporiadať a poskytnúť zainteresovaným stranám prístup k relevantným a aktuálnym informáciám, vytvorili pracovné skupiny ISTQB® na webových stránkach *www.istqb.org* odkazy, ktoré vedú na podpornú dokumentáciu a zmeny noriem. Znalosť týchto informácií nie je pri skúške základnej úrovne vyžadovaná.

## 0.10 Úroveň detailu

Úroveň detailu v týchto osnovách umožňuje vykonávať kurzy a skúšky v medzinárodne porovnateľnom rozsahu. Preto sa osnovy skladajú z:

- všeobecných cieľov výučby popisujúcich zámer základnej úrovne
- zoznamu pojmov (kľúčových slov), ktoré si študenti musia byť schopní vybaviť
- študijných cieľov pre každú oblasť znalostí popisujúcich výsledok učenia, ktoré sa má dosiahnuť
- popisu kľúčových konceptov vrátane odkazov na uznávané zdroje.

Obsahom osnov nie je vyčerpávajúci popis všetkých znalostí v oblasti testovania softvéru, ale odráža úroveň detailu, ktorá má byť pokrytá v rámci vzdelávacích kurzov pre základnú úroveň. Obsah sa zameriava na koncepciu a techniky testovania, ktoré je možné použiť pre všetky softvérové projekty nezávisle na zvolenom SDLC (software development lifecycle, životný cyklus vývoja softvéru).

## 0.11 Usporiadanie osnov

Súčasťou učebných osnov je celkom šesť kapitol, ktorých obsah môže byť súčasťou skúšky. Najvyššia úroveň nadpisu pre každú kapitolu uvádza celkovú dobu výučby pre danú kapitolu, časovanie pre nižšie úrovne kapitoly nie je definované. Celkom je tak pri akreditovaných vzdelávacích kurzoch vyžadovaný rozsah najmenej 1135 minút (18 hodín a 55 minút) rozdelených do šiestich kapitol a to takto:

- Kapitola 1: Základy testovania (180 minút)
- Študenti sa zoznámia so základnými princípmi súvisiacimi s testovaním, dôvodmi, prečo je testovanie vyžadované a s popisom cieľov testovania.
- Študenti porozumejú pojmom proces testovania, hlavné testovacie činnosti a testvér.
- Študenti sa zoznámia so základnými zručnosťami potrebnými na testovanie.
- Kapitola 2: Testovanie v priebehu životného cyklu vývoja softvéru (130 minút)
- Študenti porozumejú tomu, ako je testovanie začlenené do rôznych metodík vývoja.
- Študenti sa zoznámia s konceptmi prístupov iniciovaných testami (test-first) a s metodikami DevOps.
- Študenti sa zoznámia s rôznymi úrovňami testov, typmi testovania a testovaním počas údržby.
- Kapitola 3: Statické testovanie (80 minút)
- Študenti sa zoznámia so základmi statického testovania, s procesom spätnej väzby a revízie.
- Kapitola 4: Testovacia analýza a návrh testov (390 minút)
- Študenti sa naučia aplikovať techniky čiernej skrinky, bielej skrinky a techniky testovania založené na skúsenostiach pri odvodzovaní testovacích prípadov z rôznych softvérových pracovných produktov.
- Študenti sa zoznámia s prístupom k testovaniu založeným na spolupráci.
- Kapitola 5: Manažment testovania (335 minút)
- Študenti sa naučia, ako všeobecne plánovať testy a ako odhadnúť úsilie potrebné na testovanie.
- Študenti sa zoznámia s tým, ako môžu riziká ovplyvniť testovanie.
- Študenti sa naučia monitorovať a riadiť testovacie aktivity.
- Študenti sa zoznámia s tým, ako konfiguračný manažment podporuje testovanie.
- Študenti sa naučia reportovať defekty jasným a zrozumiteľným spôsobom.
- Kapitola 6: Testovacie nástroje (20 minút)
- Študenti sa naučia klasifikovať nástroje a porozumieť rizikám a prínosom automatizácie testovania.

## 0.12 Použité skratky

Nasledujúca tabuľka uvádza zoznam skratiek použitých v týchto osnovách, zároveň je význam skratky uvedený pri jej prvom výskyte. Definíciu niektorých skratiek je možné nájsť aj v slovníku ISTQB® na adrese *glossary.istqb.org*.

| Skratka | Význam | Slovenský význam |
|---|---|---|
| ISTQB | International Software Qualifications Board | Medzinárodný výbor pre testovanie softvéru |
| CTFL | Certified Tester Foundation Level | Certifikovaný tester základnej úrovne |
| SDLC | software development lifecycle | životný cyklus vývoja softvéru |
| QA | quality assurance | zabezpečenie kvality |
| QC | quality control | kontrola kvality |
| ATDD | acceptance test driven development | vývoj riadený akceptačnými testami |
| BDD | behavior driven development | vývoj riadený správaním |
| DDD | domain driven development | vývoj riadený doménou |
| XP | extreme programming | Extrémne programovanie |
| FDD | feature driven development | vývoj riadený funkcionalitou (FDD - Feature-driven development) |
| TDD | test driven development | vývoj riadený testami |
| CI | continuous integration | priebežná integrácia |
| CD | continuous delivery | priebežné dodávanie |
| EP | equivalence partitioning | rozdelenie na triedy ekvivalencie |
| BVA | boundary value analysis | analýza hraničných hodnôt |
| API | application programming interface | aplikačné programové rozhranie (API) |
| E2E | end-to-end | neprekladá sa, významovo ide o testovanie od začiatku do konca |
| DDP | defect density percentage | hustota defektov vyjadrená v percentách (podiel všetkých defektov na jednotku veľkosti objektu) |
| KPI | key performance indicator | kľúčový výkonnostný ukazovateľ (KPI) |

# 1 Základy testovania - 180 minút

**Kľúčové slová**

cieľ testovania, defekt, dokončenie testovania, chyba, implementácia testovania, koreňová príčina, kvalita, ladenie, monitorovanie testovania, návrh testov, plánovanie testovania, pokrytie, riadenie testovania, testovacia analýza, testovacia báza, testovacia podmienka, testovacia procedúra, testovacie dáta, testovací prípad, testovanie, testovaný objekt, testvér, validácia, verifikácia, vykonanie testov, výsledok testu, zabezpečenie kvality, zlyhanie

**Študijné ciele**

## 1.1 Čo je testovanie?

FL-1.1.1 (K1) Identifikovať typické ciele testovania.

FL-1.1.2 (K2) Rozlišovať testovanie od ladenia.

## 1.2 Prečo je testovanie nevyhnutné?

FL-1.2.1 (K2) Ilustrovať na príkladoch, prečo je testovanie nevyhnutné.

FL-1.2.2 (K1) Zapamätať si vzťah medzi testovaním a zabezpečením kvality.

FL-1.2.3 (K2) Rozlišovať medzi koreňovou príčinou, chybou, defektom a zlyhaním.

## 1.3 Princípy testovania

FL-1.3.1 (K2) Vysvetliť sedem princípov testovania.

## 1.4 Testovacie činnosti, testvér a roly v testovaní

FL-1.4.1 (K2) Zhrnúť rôzne testovacie činnosti a úlohy.

FL-1.4.2 (K2) Vysvetliť vplyv kontextu na proces testovania.

FL-1.4.3 (K2) Rozlišovať medzi rôznymi druhymi testvéru podporujúcimi testovacie činnosti.

FL-1.4.4 (K2) Vysvetliť hodnotu udržiavania sledovateľnosti.

FL-1.4.5 (K2) Porovnať rôzne roly v testovaní.

## 1.5 Základné zručnosti a osvedčené postupy pri testovaní

FL-1.5.1 (K2) Uviesť príklady všeobecných zručností požadovaných pre testovanie.

FL-1.5.2 (K1) Zapamätať si prínosy tímového prístupu.

FL-1.5.3 (K2) Uvedomiť si výhody a nevýhody nezávislosti testovania.

# 1.1 Čo je testovanie?

Softvérové systémy sú neoddeliteľnou súčasťou nášho každodenného života. Väčšina ľudí má skúsenosti so softvérom, ktorý fungoval inak, než očakávali. Softvér, ktorý nefunguje správne, môže viesť k mnohým problémom, napr. strate peňazí, času alebo obchodnej povesti a v extrémnych prípadoch dokonca k zraneniu alebo smrti. Testovanie softvéru hodnotí kvalitu softvéru a pomáha znižovať riziko zlyhania softvéru v prevádzke.

Testovanie softvéru je sada činností, ktorých cieľom je odhaľovanie defektov a vyhodnocovanie kvality softvérových artefaktov, ktoré sa pri testovaní označujú ako testované objekty. Bežná (a bohužiaľ mylná) predstava o testovaní je, že ide len o vykonávanie testov (t. j. spustenie softvéru a kontrola výsledkov testov). Testovanie softvéru ale zahŕňa aj ďalšie činnosti a musí byť v súlade so životným cyklom vývoja softvéru (pozri *kapitolu 2*).

Ďalšou bežnou mylnou predstavou o testovaní je, že testovanie sa zameriava výhradne na verifikáciu testovaného objektu. Testovanie ale zahŕňa nielen verifikáciu (t. j. kontrolu, či systém spĺňa špecifikované požiadavky), ale aj validáciu (teda kontrolu, či systém spĺňa potreby používateľov a ďalších zainteresovaných strán v ich prevádzkovom prostredí).

Testovanie môže byť dynamické alebo statické. Dynamické testovanie vyžaduje spustenie testovaného softvéru, zatiaľ čo statické testovanie spustenie nevyžaduje. Statické testovanie zahŕňa revízie (pozri *kapitolu 3*) a statickú analýzu. Dynamické testovanie používa rôzne techniky testovania a prístupy k testovaniu s cieľom definovať testovacie prípady (pozri *kapitolu 4*).

Testovanie nie je len technická činnosť. Rovnako je potrebné ho riadne plánovať, riadiť, odhadovať, monitorovať a kontrolovať (pozri *kapitolu 5*).

Testeri síce používajú pri testovaní nástroje (pozri *kapitolu 6*), ale je dôležité si uvedomiť, že testovanie je hlavne intelektuálna činnosť. Pri nej je potrebné, aby testeri mali špecializované znalosti, používali analytické zručnosti a aplikovali kritické a systémové myslenie ((Myers et al., 2011),(Roman, 2018)).

Ďalšie informácie o konceptoch testovania softvéru možno nájsť v norme ISO/IEC/IEEE 29119-1 (*Softwarové a systémové inženýrství – Testování softwaru – Část 1: Obecné pojmy*, 2022).

## 1.1.1 Typické ciele testovania

Typické ciele testovania sú:

- ohodnotenie pracovných produktov ako sú požiadavky, používateľské scenáre, návrhy a kód,
- vyvolanie zlyhania a nájdenie defektov,
- zabezpečenie požadovaného pokrytia testovaného objektu,
- zníženie úrovne rizika plynúce z nedostatočnej kvality softvéru,
- verifikácia, či boli splnené stanovené požiadavky,
- verifikácia, že testovaný objekt vyhovuje zmluvným, právnym a regulatórnym požiadavkám,
- poskytnutie informácií zainteresovaným stranám tak, ako základ pre kvalifikované rozhodnutia,
- vytvorenie dôvery v danú úroveň kvality testovaného objektu,

- kontrola, či je testovaný objekt kompletný a funguje podľa očakávania všetkých zainteresovaných strán.

Ciele testovania sa môžu líšiť v závislosti od kontextu, ktorý zahŕňa faktory ako je charakteristika testovaného pracovného produktu, úroveň testovania, rizík, zvoleného životného cyklu vývoja softvéru (SDLC). Okrem toho ale závisí aj od biznisového kontextu s faktormi ako štruktúra spoločnosti, konkurenčné hľadiská alebo doba uvedenia na trh (time to market).

### 1.1.2 Testovanie a ladenie

Testovanie a ladenie (debugging) sú samostatné činnosti. Testovanie môže vyvolať zlyhania, ktoré sú spôsobené defektmi v softvéri (dynamické testovanie), alebo môže priamo odhaliť defekty v testovanom objekte (statické testovanie).

Zatiaľ čo dynamické testovanie (pozri *kapitolu 4*) sa snaží vyvolať zlyhanie, ladenie sa zaoberá hľadaním príčin takých zlyhaní (t. j. defektov), ich analýzou a ich odstránením. Typický proces ladenia v tomto prípade zahŕňa:

- reprodukovanie zlyhania,
- diagnostika (nájdenie koreňovej príčiny),
- oprava príčiny.

Následné tzv. konfirmačné testovanie overuje, či došlo v dôsledku týchto opráv k vyriešeniu problému. Vykonáva ho ideálne rovnaká osoba, ktorá vykonala prvotný test. Okrem toho je následne tiež možné vykonať tzv. regresné testovanie s cieľom overiť, či opravy nespôsobujú zlyhanie v iných častiach testovaného objektu. Viac informácií o konfirmačnom a regresnom testovaní pozri *kapitolu 2.2.3*.

Statické testovanie priamo identifikuje defekt, ladenie sa zaoberá jeho odstránením. Obvykle pri ňom nie je nutné vykonávať reprodukovanie a diagnostiku, pretože statické testovanie nachádza priamo defekty a nemôže spôsobiť zlyhanie (pozri *kapitolu 3*).

## 1.2 Prečo je testovanie nevyhnutné?

Testovanie ako jeden zo spôsobov riadenia kvality pomáha pri dosiahnutí dohodnutých cieľov v rámci stanoveného rozsahu, času, kvality a rozpočtových obmedzení. K úspechu projektu neprispievajú nutne iba činnosti zaistené testovacím tímom – ktokoľvek zo zainteresovaných strán môže využiť svoje testovacie zručnosti na to, aby sa projekt úspešne priblížil k svojmu cieľu. Testovanie komponentov, systémov a súvisiacej dokumentácie pomáha identifikovať defekty v softvéri.

### 1.2.1 Ako testovanie prispieva k úspechu

Testovanie poskytuje nákladovo efektívny spôsob detekcie defektov. Tieto defekty je možné odstrániť (ladením, teda aktivitou, ktorá nespadá pod oblasť testovania), takže testovanie vlastne nepriamo prispieva ku kvalitnejším testovaným objektom.

Testovanie poskytuje prostriedky na priame hodnotenie kvality testovaného objektu v rôznych fázach SDLC. Tieto hodnotenia sa používajú ako súčasť širších aktivít manažmentu projektov a prispievajú tak k rozhodnutiu o prechode do ďalšej fázy SDLC, napríklad k rozhodnutiu o vydaní.

Testeri sú zárukou toho, že ich pochopenie potrieb používateľov je zohľadnené v priebehu celého životného cyklu vývoja. Alternatívou k využitiu testerov je zapojenie reprezentatívnej skupiny používateľov do projektu, čo ale obvykle nie je možné kvôli vysokým nákladom a nedostatočnej dostupnosti vhodných používateľov.

Okrem hodnotenia kvality testovaného objektu môže byť testovanie softvéru vyžadované aj z dôvodu splnenia zmluvných alebo zákonných požiadaviek či noriem.

### 1.2.2 Testovanie a zabezpečenie kvality

Ľudia často zamieňajú pojmy testovania a zabezpečenie kvality (QA – quality assurance), ale nie je to to isté.

Testovanie je forma riadenia kvality (QC – quality control). QC je produktovo orientovaný, nápravný prístup, ktorý sa zameriava na tie činnosti, ktoré podporujú dosiahnutie primeranej úrovne kvality. Testovanie je síce hlavnou formou QC, ale nie je jedinou. Možno využiť aj iné formálne metódy (napr. overenie modelu alebo dôkazy správnosti), prípadne simuláciu alebo prototypovanie.

QA je preventívny prístup orientovaný na procesy, ktorý sa zameriava na ich zavádzanie a zlepšovanie. Je založený na predpoklade, že pokiaľ je dodržiavaný správny proces, potom je jeho výsledkom správny produkt. QA sa vzťahuje na procesy vývoja i testovania a je zodpovednosťou všetkých osôb v projekte.

Výsledky testov sú využívané v QA aj v QC. V QC sa používajú na opravu defektov, zatiaľ čo v QA poskytujú spätnú väzbu o tom, ako dobre fungujú procesy vývoja a testovania.

### 1.2.3 Chyby, defekty, zlyhania a koreňové príčiny

Ľudia robia chyby (omyly), v dôsledku ktorých vznikajú defekty (vady, bugy), ktoré môžu viesť k zlyhaniu. Ľudské chyby vznikajú z rôznych dôvodov ako sú časová tieseň alebo zložitosť pracovných produktov, procesov, infraštruktúry a interakcií. Chyby môžu tiež vzniknúť jednoducho preto, že sú ľudia unavení alebo im chýbajú zodpovedajúce školenia.

Defekty možno nájsť v dokumentácii (napr. v špecifikácii požiadaviek alebo testovacom skripte), v zdrojovom kóde alebo v akomkoľvek podpornom artefakte, napr. v zostavenom súbore (builde). Pokiaľ nie sú zistené defekty artefaktov vzniknutých v skorších fázach SDLC, vedú často k vzniku defektných artefaktov v jeho nadväzujúcich fázach.

Ak je spustený defekt v kóde, systém nemusí robiť to, čo by mal robiť alebo môže urobiť niečo, čo by nemal. To môže (ale nemusí) spôsobiť zlyhanie. Niektoré defekty môžu viesť pri spustení k zlyhaniu vždy, iné iba za určitých okolností, niektoré dokonca nemusia zlyhanie spôsobiť nikdy.

Chyby a defekty nie sú jedinou príčinou zlyhania. Zlyhania môžu byť spôsobené aj podmienkami prostredia, príkladom môže byť defekt vo firmvéri spôsobený žiarením alebo pôsobením elektromagnetického poľa.

Koreňová príčina je primárnym dôvodom vzniku problému (napr. situácia, ktorá vedie k chybe). Koreňové príčiny možno identifikovať pomocou ich analýzy, ktorá sa obvykle vykonáva po výskyte zlyhania alebo identifikácii defektu. Predpokladá sa, že riešením koreňovej príčiny (najlepšie jej odstránením) možno zabrániť podobným zlyhaniam alebo defektom (popr. aspoň znížiť ich početnosť).

## 1.3 Princípy testovania

Počas posledných rokov bol zdokumentovaný rad princípov, ktoré poskytujú všeobecný návod aplikovateľný pre všetky druhy testovania. Tieto učebné osnovy popisujú sedem takýchto princípov.

**1. Testovanie preukazuje prítomnosť defektov, nie ich neprítomnosť.** Testovanie môže preukázať, že defekty sú v testovanom objekte prítomné, ale nemôže preukázať, že v ňom žiadne nie sú (Buxton et al., 1970). Testovanie znižuje pravdepodobnosť, že v testovanom objekte zostali neodhalené defekty, avšak pokiaľ nie sú žiadne defekty nájdené, nie je tým preukázaná jeho správnosť.

**2. Vyčerpávajúce testovanie nie je možné.** Testovanie všetkého (napr. všetkých kombinácií vstupov) nie je možné s výnimkou triviálnych prípadov (Manna et al., 1978). Namiesto snahy o vyčerpávajúce testovanie je lepšie sa zamerať na vhodné techniky testovania (pozri *kapitolu 4*), stanovenie priorít testovacích prípadov (pozri *kapitolu 5.1.5*) a testovanie založené na rizikách (pozri *kapitolu 5.2*).

**3. Včasné testovanie šetrí čas a peniaze.** Defekty, ktoré sú odstránené v ranej fáze procesu, nespôsobia následne iné defekty v odvodených (následných) pracovných produktoch. Náklady na kvalitu budú navyše znížené, pretože neskôr v SDLC dôjde k menšiemu počtu zlyhaní (Boehm, 1981). Na včasné odhalenie defektov by sa malo čo najskôr začať statické testovanie (pozri *kapitolu 3*) aj dynamické testovanie (pozri *kapitolu 4*).

**4. Zhlukovanie defektov.** Väčšinu zistených defektov obsahuje obvykle malé množstvo systémových komponentov, prípadne je malé množstvo systémových komponentov zodpovedných za väčšinu prevádzkových zlyhaní (Endres, 1975). Tento jav je ilustráciou tzv. Paretovho pravidla. Predpokladané a skutočne pozorované zhluky defektov počas testovania alebo v prevádzke sú významným vstupom pre testovanie založené na rizikách (pozri *kapitolu 5.2*).

**5. Testy sa opotrebovávajú.** Pokiaľ sa rovnaké testy opakujú neustále dookola, stávajú sa stále menej účinnými a nedochádza k odhaľovaniu nových defektov (Beizer, 1990). Aby sme sa vysporiadali s dôsledkami tohto princípu, je nutné meniť existujúce testy a testovacie dáta, príp. je potrebné vytvoriť nové testy. V niektorých prípadoch má však opakovanie rovnakých testov význam, napr. pri automatizovanom regresnom testovaní (pozri *kapitolu 2.2.3*).

**6. Testovanie je závislé od kontextu.** Neexistuje jediný univerzálne použiteľný prístup k testovaniu, testovanie sa vykonáva odlišne v rôznych kontextoch (Kaner; Falk et al., 1999).

**7. Neprítomnosť defektov je klam.** Je mylnou predstavou očakávať, že obyčajná verifikácia softvéru zaistí jeho úspech. Aj pri dôkladnom testovaní všetkých špecifikovaných požiadaviek a odstránení všetkých zistených defektov môže vzniknúť systém, ktorý nespĺňa potreby a očakávania používateľov, ktorý nepomáha dosahovať biznisové ciele zákazníka a ktorý je horší v porovnaní s inými konkurenčnými systémami. Okrem verifikácie by mala byť vykonaná aj validácia (Boehm, 1981).

## 1.4 Testovacie činnosti, testvér a roly v testovaní

Testovanie je síce závislé od kontextu, avšak existujú testovacie činnosti, bez ktorých je menej pravdepodobné, že sa dosiahnu vytýčené ciele. Skupiny takýchto činností nazývame proces testovania. Proces testovania môže byť prispôsobený danej situácii na základe rôznych faktorov. O tom, ktoré testovacie činnosti sú zahrnuté do procesu testovania, ako sú implementované a kedy k nim dôjde, sa obvykle rozhoduje v rámci plánovania testovania pre konkrétnu situáciu (pozri *kapitolu 5.1*).

Nasledujúci text popisuje všeobecné aspekty tohto procesu z hľadiska testovacích činností a úloh, vplyvu kontextu, testvéru, sledovateľnosti medzi testovacou bázou a testvérom a tiež rolami v testovaní.

Ďalšie informácie o procesoch testovania je možné nájsť v norme ISO/IEC/IEEE 29119-2 (*Softwarové a systémové inženýrství - Testování softwaru - Část 2: Testovací procesy*, 2021).

### 1.4.1 Testovacie činnosti a úlohy

Proces testovania sa obvykle skladá z hlavných skupín činností (pozri ďalej), ktoré je obvykle nutné prispôsobiť systému a projektu. Aj keď sa mnohé z týchto hlavných skupín činností môžu javiť ako logicky zoradené za sebou, často sa vykonávajú iteratívne alebo paralelne.

**Plánovanie testovania** sa skladá z definovania cieľov testovania a následného výberu takého prístupu, ktorý najlepšie dosahuje definované ciele v rámci obmedzení daných celkovým kontextom. Plánovanie testovania je ďalej vysvetlené v *kapitole 5.1*.

**Monitorovanie a riadenie testovania**. Monitorovanie testovania zahŕňa priebežnú kontrolu všetkých testovacích činností a porovnávanie skutočného stavu proti plánu, riadenie testovania sa zaoberá prijatím opatrení potrebných na dosiahnutie cieľov testovania. Monitorovanie a riadenie testovania sú ďalej vysvetlené v *kapitole 5.3*.

**Testovacia analýza** zahŕňa analýzu testovacej bázy za účelom identifikácie testovateľných funkcionalít (features) a definovania príslušných testovacích podmienok vrátane stanovenia ich priorít a súvisiacich rizík spolu s ich úrovňami (pozri *kapitolu 5.2*). Testovacia báza a testované objekty sú tiež hodnotené za účelom posúdenia ich testovateľnosti a identifikácie prípadných defektov v nich obsiahnutých. Pri testovacej analýze sú často používané techniky testovania (pozri *kapitolu 4*). Testovacia analýza určuje „Čo sa má testovať?“ v zmysle merateľných kritérií pokrytia.

**Návrh testov** zahŕňa rozpracovanie testovacích podmienok do testovacích prípadov a ďalšieho testvéru (napr. testovacích listín). Často obsahuje aj identifikáciu položiek pokrytia slúžiacich ako vodítko pre stanovenie vstupov testovacích prípadov. Pri tejto činnosti je možné použiť taktiež vhodné techniky testovania (pozri *kapitolu 4*). Návrh testov tiež zahŕňa stanovenie požiadaviek na testovacie dáta, návrh testovacieho prostredia a identifikáciu ďalšej potrebnej infraštruktúry alebo nástrojov. Návrh testov dáva odpoveď na otázku „Ako testovať?“.

**Implementácia testovania** zahŕňa vytvorenie alebo získanie testvéru nevyhnutného na vykonanie testov (napr. testovacie dáta). Testovacie prípady môžu byť usporiadané do testovacích procedúr a sú často zoskupené do testovacích sád. Okrem iného sú v rámci tejto činnosti vytvárané manuálne a automatizované testovacie skripty. Z dôvodu efektívneho vykonania testov (pozri *kapitolu 5.1.5*) sú testovacie procedúry prioritizované a usporiadané v harmonograme vykonania testov. Súčasťou činností implementácie testovania je zostavenie testovacieho prostredia a overenie jeho správneho nastavenia.

**Vykonanie testov** zahŕňa spustenie testov v súlade s harmonogramom vykonania testov (ako čiastkových behov testov, test runs). Vykonanie testov môže byť manuálne alebo automatizované a môže mať mnoho foriem vrátane tzv. kontinuálneho alebo párového testovania. V rámci tejto činnosti sú porovnané skutočné výsledky testov s očakávanými a zaznamenané ich výsledky. Anomálie sú analyzované s cieľom identifikácie ich pravdepodobnej príčiny. Táto analýza umožňuje reportovať anomálie na základe pozorovaných zlyhaní (pozri *kapitolu 5.5*).

K činnostiam **dokončenia testovania** obvykle dochádza pri dosiahnutí míľnikov projektu (napr. vydanie, koniec iterácie, dokončenie úrovne testovania) a vykonáva sa pre všetky nevyriešené defekty, požiadavky na zmeny alebo vytvorené položky produktového backlogu. Akýkoľvek testvér, ktorý môže

byť v budúcnosti užitočný, je identifikovaný a archivovaný, prípadne je odovzdaný príslušným tímom. Testovacie prostredie sa uvedie do dohodnutého stavu (napr. je vypnuté). Testovacie činnosti sú analyzované s cieľom identifikácie získaných poznatkov a vylepšení pre prípadné budúce iterácie, vydania alebo aj iné projekty (pozri *kapitolu 2.1.6*). Je vytvorená správa o dokončení testovania, ktorá je komunikovaná príslušným zainteresovaným stranám.

### 1.4.2 Proces testovania v súvislostiach

Testovanie neprebieha izolovane. Testovacie činnosti sú neoddeliteľnou súčasťou procesov vývoja vykonávaných v rámci organizácie. Finálnym cieľom testovania je (okrem iného) pomôcť splniť biznisové potreby zainteresovaných strán, ktoré tiež proces testovania financujú. Spôsob, akým sa testovanie vykonáva, bude preto závisieť od mnohých faktorov, napríklad:

- zainteresované strany (ich potreby, očakávania, požiadavky, ochota spolupracovať),
- členovia tímu (ich zručnosti, znalosti, úroveň skúseností, dostupnosť, potreby školenia),
- sféra biznisu (kritickosť testovaného objektu, identifikované riziká, potreby trhu, konkrétne právne predpisy),
- technické faktory (typ softvéru, architektúra produktu, použité technológie),
- obmedzenie projektu (rozsah, čas, rozpočet, zdroje),
- organizačné faktory (organizačná štruktúra, existujúce politiky, používané postupy),
- životný cyklus vývoja softvéru (technické postupy, metódy vývoja atď.),
- nástroje (dostupnosť, použiteľnosť, zhoda s predpismi).

Tieto faktory budú mať vplyv na mnoho aspektov súvisiacich s testovaním vrátane stratégie testovania, použitých techník testovania, stupňa automatizácie testovania, požadovanej úrovne pokrytia, úrovne podrobnosti dokumentácie testovania, reportovania, atď.

### 1.4.3 Testvér

Testvér je definovaný ako výstupný pracovný produkt z testovacích činností popísaných v *kapitole 1.4.1*. V tom, ako rôzne organizácie vytvárajú, pomenovávajú, organizujú a spravujú také pracovné produkty, existujú však značné rozdiely. Pre podporu konzistencie a integrity testvéru je dôležité zaistenie dobrého konfiguračného manažmentu (pozri *kapitolu 5.4*).

Nasledujúci zoznam uvádza niektoré (ale nie všetky) také typické výstupné pracovné produkty (testvér):

- **Pracovné produkty z plánovania testovania:** plán testovania, harmonogram testovania, register rizík a vstupné a výstupné kritériá (pozri *kapitolu 5.1*). Register rizík je zoznam rizík spolu s ich pravdepodobnosťou, dopadom a informáciami o ich možnom zmiernení (risk mitigation), pozri *kapitolu 5.2*. Harmonogram testovania, register rizík, vstupné a výstupné kritériá sú často súčasťou plánu testovania.
- **Pracovné produkty z monitorovania a riadenia testovania:** správy o postupe testovania (pozri *kapitolu 5.3.2*), pokyny a nevyhnutné nápravné opatrenia (pozri *kapitolu 5.3*) a informácie o rizikách (pozri *kapitolu 5.2*).
- **Pracovné produkty z testovacej analýzy:** (prioritizované) testovacie podmienky (napr. akceptačné kritériá, pozri *kapitolu 4.5.2*) a defekty v testovacej báze (pokiaľ ešte neboli opravené).

- **Pracovné produkty z návrhu testov:** (prioritizované) testovacie prípady, testovacie listiny, položky pokrytia, požiadavky na testovacie dáta a požiadavky na testovacie prostredie.
- **Pracovné produkty z implementácie testovania:** testovacie procedúry, automatizované testovacie skripty, testovacie sady, testovacie dáta, harmonogram prevedenia testov a prvky testovacieho prostredia. Medzi príklady položiek testovacieho prostredia patria nástavce, ovládače, simulátory a virtualizácia služieb.
- **Pracovné produkty z vykonania testov:** protokoly testov (test logs) a reporty defektov (pozri *kapitolu 5.5*).
- **Pracovné produkty z dokončenia testovania:** správa o dokončení testovania (pozri *kapitolu 5.3.2*), akčné body pre zlepšenie budúcich projektov alebo iterácií, zdokumentované získané poznatky (lessons learned) a zmenové požiadavky (napr. ako položky produktového backlogu).

### 1.4.4 Sledovateľnosť medzi testovacou bázou a pracovnými produktmi z testovania

Aby bolo možné efektívne monitorovať a riadiť testovanie, je dôležité vytvoriť a následne udržiavať v priebehu celého procesu testovania sledovateľnosť medzi každou položkou testovacej bázy, výstupnými pracovnými produktmi vztiahnutými k tejto položke (testvérom – napr. testovacie podmienky, riziká alebo testovacie prípady), výsledky testov a nájdenými defektmi.

Presná sledovateľnosť umožňuje vyhodnocovať pokrytie, takže je veľmi užitočné, pokiaľ sú v testovacej báze pre toto pokrytie definované jeho merateľné kritériá. Kritériá pokrytia môžu predstavovať jeden z kľúčových ukazovateľov výkonnosti (KPI – key performance indicator) pre riadenie testovacích činností, ktoré ukazujú na mieru dosiahnutia cieľov testovania (pozri *kapitolu 1.1.1*). Napríklad:

- Sledovateľnosť medzi testovacími prípadmi a požiadavkami pomôže overiť, či sú požiadavky pokryté testovacími prípadmi.
- Sledovateľnosť medzi výsledkami testov a rizikami možno použiť na vyhodnotenie úrovne zvyškového rizika testovaného objektu.

Okrem vyhodnotenia pokrytia umožňuje dobrá sledovateľnosť určiť vplyv zmien, uľahčuje vykonávanie auditov testovania a pomáha plniť kritériá zásad správneho riadenia v IT. Dobrá sledovateľnosť tiež zlepšuje zrozumiteľnosť správ o postupe testovania a správ o dokončení testovania tým, že ukazuje stav položiek testovacej bázy. To môže tiež prispieť k zrozumiteľnej komunikácii so zainteresovanými stranami o technických aspektoch testovania. Sledovateľnosť poskytuje informácie pre posúdenie kvality produktu, spôsobilosti procesov a postupe prác v projekte vzhľadom na biznisové ciele.

### 1.4.5 Roly v testovaní

V týchto učebných osnovách sú definované dve hlavné roly pri testovaní: rola manažéra testovania a rola testera. Činnosti a úlohy vykonávané týmito dvoma rolami závisia od kontextu projektu a produktu, od zručností ľudí v daných rolách a od danej organizácie.

**Manažér testovania** má celkovú zodpovednosť za proces testovania, testovací tím a úspešné riadenie testovacích činností. Zameriava sa predovšetkým na činnosti plánovania testovania, monitorovania a riadenia testovania a dokončenia testovania. Spôsob, akým sa táto rola vykonáva, sa líši v závislosti na životnom cykle vývoja softvéru, napríklad v agilnom vývoji sú niektoré úlohy manažéra testovania vykonávané celým agilným tímom. Úlohy, ktoré zahŕňajú viac tímov alebo celú organizáciu, môžu byť vykonávané manažérmi testovania mimo vývojový tím.

**Tester** preberá celkovú zodpovednosť za inžiniersky (technický) aspekt testovania. Zameriava sa predovšetkým na činnosti testovacej analýzy, návrhu testov, implementácie testovania a vykonania testov.

Obe roly môžu byť vykonávané v rôznych obdobiach rôznymi ľuďmi, napríklad rola manažéra testovania môže byť vykonávaná vedúcim tímu, samotným manažérom testovania alebo manažérom vývoja. Je tiež možné, aby jedna osoba prevzala rolu testera aj rolu manažéra testovania súčasne.

## 1.5 Základné zručnosti a osvedčené postupy pri testovaní

Zručnosť je schopnosť robiť niečo dobre a vychádza z niečích znalostí, praxe a schopností. Aby mohli testeri dobre vykonávať svoju prácu, potrebujú mať určité základné zručnosti, okrem iného by mali byť dobrými tímovými hráčmi a mali by byť schopní pracovať na rôznych úrovniach nezávislosti testovania.

### 1.5.1 Všeobecné zručnosti potrebné na testovanie

Nasledujúce zručnosti sú síce všeobecné, ale pre testerov obzvlášť dôležité:

- Znalosť testovania (s cieľom zvyšovania efektivity testovania, napr. pomocou techník testovania).
- Dôkladnosť, starostlivosť, zvedavosť, dôraz na detail, schopnosť pracovať metodicky (s cieľom identifikácie defektov, a to najmä tých, ktoré sa dajú len ťažko nájsť).
- Dobré komunikačné zručnosti, aktívne počúvanie, schopnosť byť tímovým hráčom (s cieľom efektívne komunikovať so všetkými zainteresovanými stranami, odovzdávať informácie ostatným, byť pochopený, reportovať defekty a diskutovať o nich).
- Analytické myslenie, kritické myslenie, kreativita (s cieľom zvyšovania efektivity testovania).
- Technické znalosti (s cieľom zvyšovania efektivity testovania, napr. použitím vhodných testovacích nástrojov).
- Znalosť domény (s cieľom mať schopnosť porozumieť koncovým používateľom alebo obchodným zástupcom a komunikovať s nimi).

Bežnou ľudskou vlastnosťou je obviňovanie nositeľov zlých správ a testeri sú často nositeľmi takých zlých správ. Z tohto dôvodu sú pre testerov kľúčové komunikačné schopnosti – oznamovanie výsledkov testov môže byť totiž vnímané ako kritika produktu a jeho autora.

Termín konfirmačné skreslenie (confirmation bias) popisuje tendenciu človeka ťažko prijímať informácie, ktoré sú v rozpore s jeho súčasnými názormi. Ide o psychologický faktor, v dôsledku ktorého môžu niektorí ľudia vnímať testovanie ako deštruktívnu aktivitu, hoci významne prispieva k úspechu projektu a kvalite produktu. Pri snahe znížiť tieto vnímania by informácie o defektoch a zlyhaniach mali byť komunikované konštruktívne a bez emócií.

### 1.5.2 Tímový prístup

Jednou z dôležitých zručností testerov je schopnosť efektívne pracovať v tíme a pozitívne prispievať k tímovým cieľom. Tzv. tímový prístup (whole team approach, technika vychádzajúca z extrémneho programovania, pozri *kapitolu 2.1*) je postavený na tejto zručnosti.

Pri tímovom prístupe môže každý člen tímu s potrebnými znalosťami a zručnosťami vykonávať akúkoľvek úlohu a každý člen tímu je zodpovedný za kvalitu. Členovia tímu zdieľajú rovnaký pracovný priestor,

pretože spoločné umiestnenie v jednom fyzickom alebo virtuálnom priestore uľahčuje vzájomnú komunikáciu a interakciu. Tímový prístup zvyšuje tímovú dynamiku, zlepšuje komunikáciu a spoluprácu v rámci tímu a vytvára synergiu tým, že umožňuje využitie rôznych zručností v tíme v prospech projektu.

Ak má byť dosiahnutá požadovaná úroveň kvality, musia testeri úzko spolupracovať s ostatnými členmi tímu. To zahŕňa (okrem iného) spoluprácu s obchodnými zástupcami (ktorí môžu napr. pomôcť s vytvorením vhodných akceptačných testov) alebo s vývojármi (napr. s cieľom dosahovať spoločné dohody o stratégii testovania a o prístupoch k automatizácii testov). Testeri môžu tiež odovzdávať znalosť testovania ostatným členom tímu a ovplyvňovať vývoj produktu.

Tímový prístup nemusí byť v kontexte projektu vždy vhodnou metodikou. Typickým príkladom je testovanie bezpečnostne kritických scenárov, kedy je nutné zachovať vysokú úroveň nezávislosti testovania.

### 1.5.3 Nezávislosť testovania

Určitá miera nezávislosti robí testerov efektívnejšími pri hľadaní defektov, čo je spôsobené rozdielmi medzi kognitívnym skreslením autorov (vývojárov) a testerov (Salman, 2016). Nezávislosť však nie je náhrada za dobrú znalosť systému – aj vývojári môžu efektívne nájsť mnoho defektov vo svojom vlastnom kóde.

Pracovné produkty môžu byť testované ich autorom (nulová nezávislosť), kolegami autora z rovnakého tímu (malá nezávislosť), testermi mimo tímu autora, ale v rámci organizácie (vysoká nezávislosť) alebo testermi mimo organizácie (veľmi vysoká nezávislosť). Vo väčšine projektov je zvyčajne najlepšie vykonávať testovanie s viacerými úrovňami nezávislosti (napr. vývojári vykonávajúci testovanie komponentov a integračné testovanie komponentov, testovací tím vykonávajúci systémové testovanie a systémové integračné testovanie a obchodní zástupcovia vykonávajúci akceptačné testovanie).

Hlavným prínosom nezávislosti testovania je to, že nezávislí testeri dokážu pravdepodobnejšie rozpoznať rôzne druhy zlyhaní a defektov v porovnaní s vývojármi, a to z dôvodu ich odlišného zázemia, technického nadhľadu a predsudkov. Nezávislí testeri môžu navyše verifikovať, napadnúť či vyvrátiť predpoklady, ktoré mali zainteresované strany v čase prípravy špecifikácie či implementácie systému.

Existujú však aj niektoré nevýhody. Nezávislí testeri môžu byť izolovaní od vývojového tímu, čo môže viesť k nedostatočnej spolupráci, problémom v komunikácii alebo nepriateľskému vzťahu s vývojovým tímom. Vývojári môžu stratiť pocit zodpovednosti za kvalitu. Nezávislí testeri môžu byť úzkym miestom vo vývojovom procese alebo môžu byť vinení za oneskorenie pri vydávaní produktu.

# 2 Testovanie v priebehu životného cyklu vývoja softvéru – 130 minút

**Kľúčové slová**

akceptačné testovanie, funkcionálne testovanie, integračné testovanie, integračné testovanie komponentov, konfirmačné testovanie, nefunkcionálne testovanie, regresné testovanie, shift-left, systémové integračné testovanie, systémové testovanie, testovanie bielej skrinky, testovanie komponentov, testovanie počas údržby, testovanie čiernej skrinky, typ testovania, úroveň testovania

**Študijné ciele**

## 2.1 Testovanie v kontexte životného cyklu vývoja softvéru

FL-2.1.1 (K2) Vysvetliť vplyv zvoleného životného cyklu vývoja softvéru na testovanie.

FL-2.1.2 (K1) Zapamätať si osvedčené postupy testovania aplikovateľné na všetky typy životného cyklu vývoja softvéru.

FL-2.1.3 (K1) Zapamätať si príklady vývoja iniciovaného testami (test-first).

FL-2.1.4 (K2) Zhrnúť, aký vplyv na testovanie môže mať DevOps.

FL-2.1.5 (K2) Vysvetliť prístup k testovaniu shift-left.

FL-2.1.6 (K2) Vysvetliť, ako je možné využiť retrospektívy ako mechanizmus na zlepšovanie procesov.

## 2.2 Úrovne testovania a typy testovania

FL-2.2.1 (K2) Rozlišovať medzi rôznymi úrovňami testovania.

FL-2.2.2 (K2) Rozlišovať medzi rôznymi typy testovania.

FL-2.2.3 (K2) Rozlišovať konfirmačné testovanie a regresné testovanie.

## 2.3 Testovanie počas údržby

FL-2.3.1 (K2) Zhrnúť testovanie počas údržby a jeho spúšťače.

# 2.1 Testovanie v kontexte životného cyklu vývoja softvéru

Model životného cyklu vývoja softvéru (SDLC) je zovšeobecnený proces vývoja softvéru. Určuje, ako spolu logicky i chronologicky súvisia rôzne fázy vývoja a typy činností vykonávaných v rámci tohto procesu. Medzi kategórie modelov SDLC patria sekvenčné modely vývoja (napr. vodopádový model, V-model), iteratívne modely vývoja (napr. špirálový model, prototypovanie) a inkrementálne vývojové modely (napr. Rational Unified Process).

Niektoré činnosti v rámci procesov vývoja softvéru je možné tiež popísať podrobnejšími metódami vývoja softvéru a agilnými postupmi. Medzi takéto metódy patrí vývoj riadený akceptačnými testami (ATDD – acceptance test-driven development), vývoj riadený správaním (BDD – behavior-driven development), návrh riadený doménou (DDD – domain-driven design), extrémne programovanie (XP – extreme programming), vývoj riadený funkcionalitou (FDD - feature-driven development), Kanban, Lean IT, Scrum a vývoj riadený testovaním (TDD – test-driven development).

## 2.1.1 Vplyv životného cyklu vývoja softvéru na testovanie

Aby bolo testovanie úspešné, musí byť prispôsobené SDLC. Voľba SDLC má vplyv na:

- rozsah a načasovanie testovacích činností (napr. úrovne testovania a typy testovania),
- úroveň detailu testovacej dokumentácie,
- voľbu techník testovania a prístupu k testovaniu,
- rozsah automatizácie testov,
- roly a zodpovednosti testerov.

V sekvenčných modeloch vývoja sa testeri v počiatočných fázach obvykle zúčastňujú revízií požiadaviek, testovacej analýzy a návrhu testov. Keďže je spustiteľný kód obvykle vytvorený až v neskorších fázach SDLC, nemožno v týchto počiatočných fázach používať techniky dynamického testovania.

V niektorých iteratívnych a inkrementálnych vývojových modeloch sa predpokladá, že výsledkom každej iterácie je funkčný prototyp alebo prírastok produktu. To znamená, že v každej iterácii môže byť vykonané statické aj dynamické testovanie, a to vo všetkých úrovniach testovania. Časté dodávanie takýchto prírastkov vyžaduje rýchlu spätnú väzbu a rozsiahle regresné testovanie.

Agilný vývoj softvéru (s využitím iteratívnych a inkrementálnych modelov) predpokladá, že v priebehu projektu môže dôjsť k zmene. Preto je v agilných projektoch uprednostňovaná skôr stručnejšia dokumentácia pracovných produktov, a naopak rozsiahla automatizácia testov, ktorá uľahčuje regresné testovanie. Väčšina manuálnych testov sa často vykonáva pomocou testovacích techník založených na skúsenostiach (pozri *kapitolu 4.4*), pri ktorých sa nevyžaduje vykonanie rozsiahlej testovacej analýzy a návrhu testov.

## 2.1.2 Životný cyklus vývoja softvéru a osvedčené testovacie postupy

Medzi osvedčené testovacie postupy, nezávisle na zvolenom modeli SDLC, patria:

- Ku každej vývojovej činnosti existuje zodpovedajúca testovacia činnosť, takže všetky vývojové činnosti podliehajú riadeniu kvality.

- Existujú rôzne úrovne testovania (pozri *kapitolu 2.2.1*) a každá má svoje špecifické (a niekedy aj odlišné) ciele. Tým je testovanie primerane detailné a zároveň nedochádza k nadbytočnostiam.
- Testovacia analýza a návrh testov pre danú úroveň začína počas zodpovedajúcej vývojovej fázy SDLC, takže je splnený princíp včasného testovania (pozri *kapitolu 1.3*)
- V okamihu, kedy je k dispozícii pracovná verzia (draft) ľubovoľného pracovného produktu, sú do jeho revízie zapojení testeri, čo (ako forma včasného testovania a včasnej detekcie defektov) podporuje prístup shift-left (pozri *kapitolu 2.1.5*).

### 2.1.3 Vývoj softvéru riadený testovaním

TDD, ATDD a BDD sú podobné prístupy k vývoju, keď sú testy definované ako prostriedok na riadenie vývoja. Každý z týchto prístupov je aplikáciou princípu včasného testovania (pozri *kapitolu 1.3*) a uplatňuje prístup shift-left (pozri *kapitolu 2.1.5*) – testy sú definované pred tvorbou samotného kódu. Všetky podporujú iteratívny model vývoja a sú charakterizované nasledovne:

**Vývoj riadený testami (TDD)**

- Písanie kódu je riadené pomocou testovacích prípadov (namiesto rozsiahleho návrhu softvéru) (Beck, 2003).
- Najprv sú spísané testy a až následne kód, a ten tak, aby týmto testom vyhovoval. Následne môžu (ale nemusia) byť testy a kód refaktorované.

**Vývoj riadený akceptačnými testami (ATDD)**

- Odvodzuje testy z akceptačných kritérií ako súčasť procesu návrhu systému (Gärtner, 2012).
- Daná časť aplikácie je vytvorená tak, aby vyhovovala testom napísaným pred samotným vývojom.

Viac informácií o ATDD možno nájsť v *kapitolu 4.5.3*.

**Vývoj riadený správaním (BDD)**

- Požadované správanie aplikácie je popísané testovacími prípadmi napísanými v jednoduchej forme prirodzeného jazyka zrozumiteľného pre zainteresované strany. Obvykle je využitý formát Given/When/Then (Daná podmienka/Podmienka/Akcia) (Chelimsky, 2010).
- Testovacie prípady sú následne automaticky preložené do spustiteľných testov.

Pre všetky vyššie uvedené prístupy môžu byť testy automatizované (ale nemusia) s cieľom podporiť kvalitu pri budúcich úpravách alebo refaktoringu.

### 2.1.4 DevOps a testovanie

DevOps je prístup k vývoju softvéru, ktorý je postavený na synergii medzi oddeleniami vývoja (vrátane testovania) a prevádzky s cieľom dosahovania definovaných spoločných cieľov. DevOps vyžaduje zmenu kultúry v rámci organizácie tak, aby sa medzi týmito dvoma organizačnými jednotkami preklenuli medzery a zároveň sa k nim pristupovalo s rovnakou dôležitosťou. DevOps podporuje autonómiu tímu, rýchlu spätnú väzbu, integrované sady nástrojov a použitie technických prístupov ako je kontinuálna integrácia (CI – continuous integration) a kontinuálne dodávanie (CD – continuous delivery). To umožňuje tímom vytvárať, testovať a vydávať kvalitný kód rýchlejšie prostredníctvom definovanej DevOps pipeline (Kim et al., 2016).

Z pohľadu testovania má DevOps tieto výhody:

- Poskytuje rýchlu spätnú väzbu o kvalite (nového) kódu a o nepriaznivom vplyve na doterajšiu funkčnosť systému/komponenty.
- Vďaka CI podporuje prístup shift-left pri testovaní (pozri *kapitolu 2.1.5*) tým, že motivuje vývojárov k písaniu kvalitného kódu podporovaného testovaním komponentov a statickou analýzou.
- Propaguje automatizované spracovanie (napr. CI/CD), ktoré uľahčuje vytváranie stabilných testovacích prostredí.
- Zvyšuje dôraz na nefunkcionálne charakteristiky kvality (napr. výkon alebo spoľahlivosť).
- Automatizáciou prostredníctvom pipeline znižuje potrebu opakovaného manuálneho testovania.
- Minimalizuje riziko regresie vďaka rozsahu a miere automatizovaných regresných testov.

DevOps má aj niektoré riziká a nevýhody:

- Musí byť definovaná a zavedená DevOps pipeline.
- Musia byť zavedené a udržiavané nástroje CI/CD.
- Automatizované testy vyžadujú dodatočné investície a môže byť ťažké ich vytvoriť a udržiavať.
- Hoci DevOps predpokladá vyšší rozsah automatizovaného testovania, nemožno zabúdať ani na manuálne testovanie, a to najmä z pohľadu koncového užívateľa

### 2.1.5 Prístup shift-left

Princíp včasného testovania (pozri *kapitolu 1.3*) je niekedy označovaný ako shift-left (posun doľava v zmysle časovej osi), kde je testovanie vykonávané v skorších fázach SDLC. Shift-left odporúča začať s testovacími činnosťami skôr (napr. nečakať na implementáciu kódu alebo na integráciu komponentov), ale neznamená to, že by testovanie v neskorších fázach malo byť zanedbávané.

Existujú niektoré osvedčené postupy, na ktorých je možné demonštrovať aplikáciu tohto prístupu:

- Revidovanie špecifikácií z pohľadu testovania, ktoré nachádza potenciálne defekty ako sú nejednoznačnosti, neúplnosti a nezrovnalosti.
- Písanie testovacích prípadov pred napísaním kódu a spustenie kódu v sade testovacieho vybavenia (test harness) počas vývoja kódu.
- Používanie CI alebo lepšie CD, pretože prichádza s rýchlou spätnou väzbou a automatizovanými testami komponentov, ktoré sú spoločne s kódom uložené do repozitára.
- Spúšťanie statickej analýzy zdrojového kódu pred dynamickým testovaním, alebo ako súčasť daného automatizovaného procesu.
- Vykonávanie nefunkcionálnych testov hneď na úrovni testovania komponentov (pokiaľ je to možné). Ide tiež o formu shift-left princípu, pretože obvykle sú tieto typy testovania vykonávané v neskorších fázach SDLC, keď je k dispozícii kompletný systém a zodpovedajúce testovacie prostredie.

Zavedenie shift-left prístupu môže v počiatočných fázach znamenať potrebu preškolenia tímu, zvýšenia prácnosti a/alebo nákladov, ale v neskorších fázach naopak šetrí vynaložené úsilie a/alebo náklady.

Je dôležité, aby boli všetci zástupcovia zainteresovaných strán presvedčení o užitočnosti tohto konceptu a podporovali ho.

### 2.1.6 Retrospektívy a zlepšovanie procesov

Retrospektívy (známe tiež ako po-projektové schôdzky alebo projektové retrospektívy) sa často konajú na konci projektu alebo iterácie, pri dosiahnutí míľnika alebo podľa potreby (ad-hoc). Načasovanie a organizácia retrospektív závisí od konkrétneho modelu SDLC. Účastníci týchto schôdzok (nielen testeri, ale aj napr. vývojári, architekti, vlastníci produktov, biznisoví analytici) obvykle preberajú:

- Čo bolo úspešné a malo by sa zachovať?
- Čo sa nepodarilo a dalo by sa zlepšiť?
- Ako implementovať návrhy na zlepšenie a zabezpečiť v budúcnosti trvalú úspešnosť?

Výstupy z retrospektív by mali byť zaznamenané, zvyčajne sú súčasťou súhrnnej správa z testovania (pozri *kapitolu 5.3.2*). Retrospektívy sú z hľadiska nastavenia procesu kontinuálneho zlepšovania kritické a je dôležité, aby boli všetky odporúčané opatrenia sledované (a dodržiavané).

Medzi typické prínosy retrospektív z pohľadu testovania patria:

- Zvyšovanie efektivity / účinnosti testov (napr. z dôvodu implementácie návrhov na zlepšenie procesov).
- Zvyšovanie kvality testwaru (napr. spoločnou revíziou testovacích procesov).
- Stmeľovanie tímu a vzájomné učenie (napr. ako výsledok možnosti hlásiť problémy a navrhovať zlepšenie).
- Zlepšovanie kvality testovacej bázy (napr. vyriešením nedostatkov v rozsahu a kvalite požiadaviek).
- Zlepšovanie spolupráce medzi vývojom a testovaním (napr. vďaka pravidelným revíziám a optimalizácii vzájomnej spolupráce).

## 2.2 Úrovne testovania a typy testovania

Úrovne testovania sú skupiny testovacích činností, ktoré sú organizované a riadené spoločne. Každá úroveň testovania je inštanciou procesu testovania tvorenou činnosťami vykonávanými vo vzťahu k softvéru v danej fáze vývoja, od jednotlivých komponentov až po celé systémy, alebo prípadne aj systémy systémov.

Úrovne testovania súvisia s ďalšími činnosťami v rámci životného cyklu vývoja softvéru. V sekvenčných modeloch SDLC sú úrovne testovania často definované tak, že výstupné kritériá jednej úrovne sú súčasťou vstupných kritérií ďalšej úrovne. To ale nemusí platiť v niektorých iteratívnych modeloch, kde sa úrovne testovania môžu v čase prekrývať a vývojové činnosti môžu preklenúť viac úrovní testovania.

Typy testovania sú skupiny testovacích činností súvisiace so špecifickými kvalitatívnymi charakteristikami a väčšinu týchto činností možno vykonávať v ľubovoľnej úrovni testovania.

### 2.2.1 Úrovne testovania

V týchto učebných osnovách je popísaných nasledujúcich päť úrovní testovania:

- **Testovanie komponentov** (označované aj ako jednotkové testovanie, unit testing) sa zameriava na testovanie izolovaných komponentov. Na ich vykonávanie je často potrebný špeciálny softvér, napr. rôzne sady testovacieho vybavenia (test harness) alebo frameworky pre jednotkové testovanie. Testovanie komponentov obvykle vykonávajú vývojári vo svojich vývojových prostrediach.
- **Integračné testovanie komponentov** (označované tiež ako integračné testovanie jednotiek) sa zameriava na testovanie rozhrania a interakcií medzi komponentmi. Toto testovanie je silne závislé na zvolenej integračnej stratégii (napr. zdola-hore, zhora-dole alebo tzv. veľký tresk).
- **Systémové testovanie** sa zameriava na celkové správanie a schopnosti celého systému alebo produktu, často vrátane funkcionálneho testovania komplexných (end-to-end) úloh a nefunkcionálneho testovania kvalitatívnych charakteristík. Niektoré z nich (napr. použiteľnosť) je vhodnejšie testovať na úplnom systéme vo vhodnom testovacom prostredí. Okrem iného je možné využívať aj simulácie subsystémov. Systémové testovanie vychádza zo špecifikácií systému a môže byť vykonávané nezávislým testovacím tímom.
- **Systémové integračné testovanie** sa zameriava na testovanie rozhrania medzi testovaným systémom a ďalšími systémami alebo externými službami. Pre systémové integračné testovanie je potrebné mať vhodné testovacie prostredie, pokiaľ možno podobné prevádzkovému.
- **Akceptačné testovanie** sa zameriava na preukázanie pripravenosti systému na nasadenie do produkčného prostredia a validuje, že systém spĺňa biznisové potreby užívateľa. V ideálnom prípade by akceptačné testovanie mali vykonávať koncoví používatelia. Najčastejšie formy akceptačného testovania sú: užívateľské akceptačné testovanie (UAT), prevádzkové akceptačné testovanie, zmluvné a regulačné akceptačné testovanie, alfa testovanie a beta testovanie.

Aby nedochádzalo k prekrývaniu činností pri testovaní v jednotlivých úrovniach, čiastkové úrovne testovania sa odlišujú špecifickou definíciou rôznych atribútov. Medzi najdôležitejšie atribúty patria:

- testovaný objekt,
- ciele testovania,
- testovacia báza,
- defekty a zlyhania,
- prístup a zodpovednosti.

### 2.2.2 Typy testovania

Existuje mnoho typov testovania, ktoré je možné na projektoch použiť. Tieto učebné osnovy sa zaoberajú štyrmi typmi testovania.

**Funkcionálne testovanie** overuje funkcionality, ktoré by komponent alebo systém mal vykonávať. Funkcionality predstavujú to, „čo“ by mal systém robiť. Hlavným cieľom funkcionálneho testovania je kontrola funkcionálnej úplnosti, funkcionálnej správnosti a funkcionálnej vhodnosti.

**Nefunkcionálne testovanie** vyhodnocuje nefunkcionálne charakteristiky komponentu alebo systému a dáva tak odpoveď na otázku „ako dobre sa systém chová“. Zoznam nefunkcionálnych charakteristík

kvality softvéru možno nájsť v norme ISO/IEC 25010 (*Systémové a softwarové inženýrství – Požadavky a hodnocení kvality systémů a softwaru (SQuaRE), Modely kvality systémů a softwaru*, 2011):

- výkonnostná efektivita,
- kompatibilita,
- použiteľnosť,
- spoľahlivosť,
- bezpečnosť,
- udržateľnosť,
- prenositeľnosť.

Veľa nefunkcionálnych testov je odvodených z funkcionálnych. Oba typy používajú rovnaké testovacie prípady, ale nefunkcionálne kontrolujú, či je pri danej funkcionalite splnené aj nejaké nefunkcionálne obmedzenie (napr. že je všetko vykonané v určenom čase alebo či môže byť daná funkčnosť prenesená na novú platformu). Neskoré odhalenie nefunkcionálneho defektu môže predstavovať vážnu hrozbu pre úspech projektu.

Niekedy je vhodné, aby nefunkcionálne testovanie začalo už v ranej fáze SDLC (napr. ako súčasť revízií a pri testovaní komponentov alebo systémovom testovaní) a niekedy je nutné využiť veľmi špecifické testovacie prostredie ako je napríklad laboratórium na testovanie použiteľnosti.

**Testovanie čiernej skrinky** (pozri *kapitolu 4.2*) je založené na špecifikácii a odvodzuje testy z dokumentácie testovaného objektu (teda nie zo správania samotného objektu). Hlavným cieľom testovania čiernej skrinky je kontrola správania systému podľa jeho špecifikácie.

**Testovanie bielej skrinky** (pozri *kapitolu 4.3*) je založené na štruktúre a odvodzuje testy z implementácie systému alebo z jeho vnútornej štruktúry, čo je napr. kód, architektúra, pracovné toky (workflows) alebo dátové toky. Hlavným cieľom testovania bielej skrinky je dosahovať dostatočné pokrytie testovanej štruktúry (napr. kódu).

Všetky štyri vyššie uvedené typy testovania je možné aplikovať vo všetkých úrovniach testovania, aj keď konkrétna implementácia bude v každej úrovni odlišná. Na odvodenie testovacích podmienok a testovacích prípadov pre všetky uvedené typy testovania je možné použiť rôzne techniky testovania.

### 2.2.3 Konfirmačné a regresné testovanie

Pokiaľ sa v komponente alebo systéme vykonávajú zmeny, je obvykle ich cieľom vylepšenie (pridaním novej funkcionality) alebo oprava (odstránením defektu). Následné testovanie by potom malo zahŕňať konfirmačné testovanie aj regresné testovanie.

**Konfirmačné testovanie** overuje, či bol pôvodný defekt úspešne opravený. V závislosti na riziku je možné otestovať opravenú verziu softvéru niekoľkými rôznymi spôsobmi, napr.:

- Vykonaním všetkých testovacích prípadov, ktoré predtým zlyhali kvôli defektu.
- Pridaním nových testov s cieľom pokrývať všetky zmeny potrebné na opravu defektu.

V prípadoch, keď je na opravu defektu málo času alebo finančných prostriedkov, môže byť konfirmačné testovanie obmedzené na jednoduché vykonanie krokov, ktoré by mali reprodukovať pôvodné zlyhanie spôsobené defektom a skontrolovať, či k zlyhaniu nedochádza.

**Regresné testovanie** overuje, či nedošlo k žiadnym nežiadúcim dopadom po realizovaných zmenách, vrátane opráv, ktoré už boli potvrdené konfirmačným testovaním. Tieto nežiadúce dopady by mohli ovplyvniť ako komponent, kde bola zmena vykonaná, tak aj iné komponenty v rovnakom systéme alebo dokonca iné prepojené systémy. Regresné testovanie nemusí byť obmedzené na samotný testovaný objekt, ale môže tiež súvisieť s prostredím. Odporúča sa preto pred testovaním najskôr vykonať tzv. analýzu dopadu s cieľom optimalizovať rozsah regresného testovania. Analýza dopadu ukazuje, ktoré časti softvéru by mohli byť ovplyvnené.

Sady regresných testov sa spúšťajú mnohokrát a obvykle sa ich počet zvyšuje s každou novou iteráciou alebo vydaním, preto sú vhodným kandidátom na automatizáciu testovania. Automatizácia týchto testov by mala začať už v raných fázach projektu (pozri *kapitolu 6*). Automatizované regresné testy je vhodné zaradiť do CI pipeline (napr. DevOps pipeline, pozri *kapitolu 2.1.4*). Regresné testy môžu byť podľa situácie vykonávané v rôznych úrovniach testovania.

Konfirmačné a/alebo regresné testovanie testovaného objektu je nutné použiť v tých úrovniach testovania, kde došlo k opravám defektov a/alebo sú v nich vykonané zmeny.

## 2.3 Testovanie počas údržby

Existujú rôzne druhy údržby s rôznymi cieľmi, napr. oprava, adaptácia na zmenu prostredia, zlepšenie výkonu alebo zlepšenie udržateľnosti (pozri normu ISO/IEC 14764 (*Softwarové inženýrství – Procesy životního cyklu softwaru – Údržba*, 2010)). Údržba môže byť ako plánovaná, tak neplánovaná (hotfix). Pred vykonaním zmeny je možné taktiež vykonať analýzu dopadu, ktorá pomôže pri rozhodovaní, či by zmena mala byť vykonaná, a to na základe potenciálnych dôsledkov v iných oblastiach systému. Testovanie zmien systému, ktorý je už v produkcii zahŕňa vyhodnotenie úspešnosti implementácie zmeny, ako aj kontrolu možnej regresie v nezmenených častiach systému (čo je obvykle väčšina systému).

Rozsah testovania počas údržby závisí od týchto faktorov:

- miera rizika zmeny,
- veľkosť existujúceho systému,
- rozsah zmeny.

Spúšťače (aktivačné udalosti) údržby a testovania údržby možno rozdeliť takto:

- Úpravy ako sú plánované vylepšenia (napr. v celom vydaní), opravné zmeny alebo hotfixy.
- Aktualizácia alebo migrácia prevádzkového prostredia (napríklad z jednej platformy na inú), kedy je nutné vykonať testy spojené s novým prostredím a zmeneným softvérom, prípadne testy konverzie dát, kedy sú dáta z inej aplikácie migrované do systému, ktorý je udržiavaný.
- Vyradenie aplikácie z prevádzky na konci jej životnosti. Pri vyradení aplikácie alebo systému môže byť vhodné vykonať testovanie archivácie dát, a to najmä pre také dáta, pre ktoré sú požadované dlhé lehoty archivácie. Pre prípad, kedy by bolo počas doby archivácie nutné načítať niektoré archivované dáta, je možné otestovať aj procesy obnovenia a načítania (po archivácii).

# 3 Statické testovanie – 80 minút

**Kľúčové slová**

anomália, dynamické testovanie, formálna revízia, inšpekcia, neformálna revízia, predvedenie (walkthrough), revízia, statická analýza, statické testovanie, technická revízia

**Študijné ciele**

## 3.1 Základy statického testovania

FL-3.1.1 (K1) Určiť typy produktov, ktoré môžu byť otestované s použitím rôznych techník statického testovania.

FL-3.1.2 (K2) Vysvetliť hodnotu statického testovania.

FL-3.1.3 (K2) Porovnať a uviesť odlišnosti statického a dynamického testovania.

## 3.2 Proces spätnej väzby a revízie

FL-3.2.1 (K1) Identifikovať prínosy včasnej a častej spätnej väzby od zainteresovaných strán.

FL-3.2.2 (K2) Zhrnúť činnosti procesu revízie.

FL-3.2.3 (K1) Zapamätať si, ktoré zodpovednosti pri vykonávaní revízií sú priradené jednotlivým rolám.

FL-3.2.4 (K2) Porovnať a uviesť odlišnosti rôznych typov revízií.

FL-3.2.5 (K1) Zapamätať si faktory, ktoré prispievajú k úspešnej revízii.

# 3.1 Základy statického testovania

Na rozdiel od dynamického testovania nie je pri statickom testovaní nutné testovaný softvér spúšťať. Kód, špecifikácia procesu, špecifikácia architektúry systému alebo iné pracovné produkty sú hodnotené a preverované buď manuálne (napr. revíziami) alebo pomocou nástroja (napr. statická analýza). Medzi ciele testovania patrí zlepšovanie kvality, odhaľovanie defektov a posudzovanie vlastností ako je čitateľnosť, úplnosť, správnosť, testovateľnosť a konzistencia. Statické testovanie je možné používať ako na verifikáciu, tak na validáciu.

Testeri, zástupcovia biznisu a vývojári počas popisu príkladov (napr. pri využití techniky špecifikácie pomocou príkladov), písaní používateľských scenárov a spresňovaní backlogu (backlog refinement) spolupracujú tak, aby používateľské scenáre a súvisiace pracovné produkty spĺňali definované kritériá, napr. definície pripravenosti (Definition of Ready) (pozri *kapitolu 5.1.3*). Techniky revízií možno použiť na zabezpečenie toho, aby používateľské scenáre boli úplné a zrozumiteľné a obsahovali testovateľné akceptačné kritériá. Tým, že testeri kladú správne otázky vlastne skúmajú, konfrontujú a pomáhajú vylepšovať navrhované používateľské scenáre.

Statická analýza môže upozorniť na problémy ešte pred vykonaním dynamického testovania, pričom často vyžaduje menšie úsilie – nevyžaduje totiž tvorbu testovacích prípadov a je možné u nej využiť rôzne nástroje (pozri *kapitolu 6*). Statická analýza je často začlenená do nástrojov priebežnej integrácie (pozri *kapitolu 2.1.4*). Aj keď sa z veľkej časti používa na odhaľovanie špecifických defektov kódu, možno ju tiež použiť na vyhodnotenie udržateľnosti a bezpečnosti (security). Ďalšími príkladmi nástrojov statickej analýzy sú nástroje na kontrolu pravopisu a čitateľnosti.

## 3.1.1 Pracovné produkty, ktoré môžu byť preverené statickým testovaním

Takmer každý pracovný produkt môže byť preskúmaný pomocou statického testovania. Medzi také produkty patria napríklad špecifikácie požiadaviek, zdrojový kód, plány testovania, testovacie prípady, položky produktového backlogu, testovacie listiny, projektová dokumentácia, zmluvy alebo modely.

**Revíziu** je možné použiť na akýkoľvek pracovný produkt, ktorý je možné prečítať a porozumieť mu.

Pre **statickú analýzu** je nutné, aby mali skúmané pracovné produkty formálnu štruktúru, voči ktorej môžu byť kontrolované (napr. modely, kód alebo text s formálnou syntaxou).

Pracovné produkty, ktoré nie sú vhodné na statické testovanie, sú obvykle tie, ktoré sú ťažko interpretovateľné ľuďmi a súčasne by nemali byť analyzované nástrojmi (napr. spustiteľný kód tretej strany, ktorý z právnych dôvodov nemožno analyzovať s využitím nástrojov).

## 3.1.2 Prínosy statického testovania

Statické testovanie môže odhaliť defekty v najskorších fázach SDLC, čím napĺňa princíp včasného testovania (pozri *kapitolu 1.3*). Môže tiež odhaliť defekty, ktoré nemožno zistiť dynamickým testovaním (napr. nedosiahnuteľný kód, chybne implementované návrhové vzory, defekty v nespustiteľných pracovných produktoch).

Statické testovanie vytvára dôveru v dané pracovné produkty a umožňuje vyhodnotiť ich kvalitu. Verifikáciou zdokumentovaných požiadaviek môžu zainteresované strany tiež zabezpečiť, že tieto požiadavky odrážajú ich skutočné potreby. Vzhľadom na to, že statické testovanie môže byť vykonávané v raných fázach SDLC, je možné nastaviť pravidlá jednotného chápania (napr. požiadaviek), čím sa tiež

zlepší komunikácia medzi zainteresovanými stranami. Z tohto dôvodu sa odporúča zapojiť do statického testovania rôzne typy účastníkov.

Aj keď môže byť vykonanie revízií nákladné, celkové náklady na projekt sú obvykle oveľa nižšie, než keď sa revízia vôbec nevykonáva. Dôvodom je to, že neskôr v projekte nie je potrebné vynaložiť toľko času a úsilia na opravu defektov.

Niektoré defekty v kóde je možné odhaliť pomocou statickej analýzy oveľa efektívnejšie ako pri dynamickom testovaní, čo obvykle vedie k ich menšiemu počtu a tým aj nižšej celkovej náročnosti pri vývoji.

### 3.1.3 Rozdiely medzi statickým a dynamickým testovaním

Statické a dynamické testovacie postupy sa vzájomne dopĺňajú. Majú podobné ciele (napr. odhaľovanie defektov v pracovných produktoch, pozri *kapitolu 1.1.1*), ale existujú medzi nimi aj určité rozdiely, napríklad:

- Ako statické, tak aj dynamické testovanie (vrátane analýzy zlyhania) môže viesť k odhaleniu defektu, avšak existujú niektoré typy defektov, ktoré možno nájsť buď len statickým alebo iba dynamickým testovaním.
- Statické testovanie nájde defekty priamo, zatiaľ čo dynamické testovanie hľadá zlyhanie, z ktorých sú príslušné defekty určené následnou analýzou.
- Statické testovanie môže ľahšie odhaľovať defekty ťažko dosiahnuteľné pomocou dynamického testovania alebo také, ktoré sú v kóde v častiach spúšťaných len zriedka.
- Dynamické testovanie je možné použiť iba na spustiteľné pracovné produkty, statické testovanie aj na nespustiteľné.
- Statické testovanie je možné použiť na meranie kvalitatívnych charakteristík (napr. udržteľnosti) nezávislých na spustení kódu, dynamické testovanie iba na tie, ktoré sú na spustení kódu závislé (napr. výkonnostná efektivita).

Medzi typické defekty odhaliteľné ľahšie a lacnejšie statickým testovaním patria:

- defekty v požiadavkách (napr. nezrovnalosti, nejednoznačnosti, rozpory, opomenutia, nepresnosti a duplicity),
- defekty v návrhu (napr. neefektívne databázové štruktúry, zlá modularita),
- niektoré typy programovacích defektov (napr. premenné s nedefinovanými hodnotami, nedeklarované premenné, nedosiahnuteľný alebo duplicitný kód, nadmerná zložitosť kódu),
- odchýlky od noriem (napr. nedostatočné dodržiavanie konvencií pre programovanie),
- nesprávne špecifikácie rozhrania (napr. rozdielny počet, typ alebo poradie parametrov),
- niektoré bezpečnostné zraniteľnosti (napr. pretečenie vyrovnávacej pamäte),
- chýbajúce časti alebo nepresnosti v pokrytí testovacej bázy (napr. chýbajúce testy pre niektoré akceptačné kritérium).

# 3.2 Proces spätnej väzby a revízie

## 3.2.1 Prínosy včasnej a častej spätnej väzby od zainteresovaných strán

Včasná a častá spätná väzba pomáha s odhalením potenciálnych problémov s kvalitou. Pokiaľ sa zainteresované strany zapájajú do vývoja nedostatočne, vyvíjaný produkt nemusí spĺňať ich pôvodné alebo súčasné predstavy. Pokiaľ nie je tím schopný dodať to, čo zainteresovaná strana chce, hrozia neočakávané náklady na prepracovanie, zmeškané termíny, vzájomné obviňovanie, a dokonca môže dôjsť k úplnému zlyhaniu projektu.

Častá spätná väzba zainteresovaných strán počas SDLC môže zabrániť nedorozumeniam pri definícii požiadaviek a zabezpečiť, aby zmeny požiadaviek boli správne a včas pochopené a implementované. To pomáha vývojovému tímu lepšie porozumieť tomu, čo vyvíja. Umožňuje im tiež zamerať sa na tie funkcionality, ktoré prinášajú zainteresovaným stranám najvyššiu pridanú hodnotu, a ktoré majú najviac pozitívny vplyv na zistené riziká.

## 3.2.2 Činnosti procesu revízie

Norma ISO/IEC 20246 (*Softwarové a systémové inženýrství – revize pracovních produktů*, 2017) definuje všeobecný proces revízie a popisuje štruktúrovaný, ale flexibilný rámec, z ktorého môže byť daný proces revízie prispôsobený konkrétnej situácii. Čím vyšší je požadovaný stupeň formálnosti revízie, tým vyšší je počet úloh počas rôznych činností.

Mnohé pracovné produkty sú tak rozsiahle, že ich nemožno pokryť iba jedinou revíziou a proces revízie môže byť vykonaný opakovane.

Činnosti procesu revízie sú:

- **Plánovanie**. Počas fázy plánovania sa stanoví rozsah revízie obsahujúci definíciu účelu a revidovaného pracovného produktu, kvalitatívne charakteristiky pre vyhodnotenie, oblasti záujmu, výstupné kritériá, doplňujúce informácie (napr. normy), potrebné úsilie a časový rámec celej revízie.
- **Začatie revízie**. Počas začatia revízie je cieľom zabezpečiť, aby bolo všetko (vrátane všetkých zainteresovaných strán) pripravené na revíziu. To okrem iného vyžaduje, aby mal každý účastník prístup k revidovanému pracovnému produktu, rozumel svojej úlohe a zodpovednostiam a dostal všetko potrebné na vykonanie revízie.
- **Individuálna revízia**. Každý revidujúci reviduje definovaný pracovný produkt individuálne s cieľom posúdiť jeho kvalitu a identifikovať anomálie, odporúčania a otázky pomocou jednej alebo viacerých techník revízie (napr. revízia založená na kontrolnom zozname, revízia podľa scenárov – pozri normu ISO/IEC 20246 (*Softwarové a systémové inženýrství – revize pracovních produktů*, 2017)). Revidujúci zaznamenáva všetky zistené anomálie, odporúčania a otázky.
- **Komunikácia a analýza**. Nie každá anomália identifikovaná počas revízie musí byť nutne defekt. Preto je nutné všetky takéto anomálie analyzovať a prediskutovať a pri každej by sa malo rozhodnúť o jej stave, vlastníctve a požadovaných opatreniach. To sa zvyčajne vykonáva pri revíznej schôdzke, počas ktorej účastníci tiež diskutujú o úrovni kvality každého revidovaného pracovného produktu a aké následné opatrenia sú vyžadované. Na uzatvorenie opatrení môže byť vyžadovaná ďalšia revízia.
- **Opravy a reportovanie**. Aby bolo možné vykonať nápravu, mal by byť pre každý defekt vytvorený report o defekte. Pri splnení daných výstupných kritérií môže byť pracovný produkt akceptovaný a

výsledky revízie sú podané v správe.

### 3.2.3 Roly a zodpovednosti pri revíziách

Revíziu môžu vykonávať rôzni ľudia a môžu zastávať rôzne roly. Hlavné roly a ich povinnosti sú:

- **Manažér** – rozhoduje o tom, čo má byť revidované a aké zdroje budú využité (vrátane ľudí a času).
- **Autor** – vytvára a opravuje revidovaný pracovný produkt.
- **Moderátor** (niekedy aj facilitátor) – zaisťuje efektívny priebeh revíznych schôdzok vrátane využitia mediácie. Okrem iného dohliada na dodržanie časového rámca a vytvorenie bezpečného prostredia, v ktorom môže každý slobodne vyjadriť svoj názor.
- **Zapisovateľ** – zhromažďuje anomálie od revidujúcich a zaznamenáva ďalšie informácie z revízie ako napr. rôzne rozhodnutia alebo nové anomálie zistené počas revízneho stretnutia.
- **Revidujúci** – vykonáva revíziu. Revidujúcim môže byť niekto, kto pracuje na projekte alebo je odborníkom na danú problematiku alebo ktorákoľvek osoba zo zainteresovaných strán.
- **Vedúci revízie** – preberá celkovú zodpovednosť za revíziu (napr. rozhodovanie o tom, kto do nej bude zapojený) a organizáciu toho, kedy a kde sa bude revízia konať.

Popis ďalších možných rolí možno nájsť v norme ISO/IEC 20246 (*Softwarové a systémové inženýrství – revize pracovních produktů*, 2017).

### 3.2.4 Typy revízií

Existuje mnoho typov revízií, od neformálnych až po veľmi formálne. Požadovaná úroveň formálnosti závisí od faktorov ako je použitý SDLC, vyspelosť procesu vývoja, kritickosť a zložitosť revidovaného pracovného produktu, právne alebo regulatórne požiadavky a potreba doloženia záznamov pre prípadný audit. Rovnaký pracovný produkt môže byť revidovaný rôznymi typmi revízií, napr. najprv neformálne a neskôr formálnejšie.

Výber správneho typu revízie je kľúčový pre dosiahnutie požadovaných cieľov revízie (pozri *kapitolu 3.2.5*). Výber je ale založený aj na ďalších faktoroch ako sú potreby projektu, dostupné zdroje, typ pracovného produktu, typ rizika, biznisová doména a firemná kultúra.

Medzi bežne používané typy revízií patria:

- **Neformálna revízia**. Neformálne revízie sa neriadia definovaným procesom a nevyžadujú formálny dokumentovaný výstup. Hlavným cieľom je odhaľovanie anomálií.
- **Predvedenie (walkthrough)**. Predvedenie (vedené autorom) môže slúžiť mnohým cieľom ako je hodnotenie kvality a budovanie dôvery v pracovný produkt, vzdelávanie revidujúcich, dosiahnutie dohody, generovanie nových nápadov, motivácia a podpora autora s cieľom zlepšovať pracovný produkt a odhaľovať anomálie. Revidujúci môžu (ale nemusia) pred predvedením vykonať individuálnu revíziu.
- **Technická revízia**. Technickú revíziu vykonávajú odborne kvalifikovaní revidujúci a vedie ju moderátor. Cieľom technickej revízie je primárne dosiahnuť zhodu a urobiť rozhodnutia týkajúce sa nejakého technického problému, ale tiež odhaliť anomálie, vyhodnotiť kvalitu, vybudovať dôveru v pracovný produkt, generovať nové nápady, motivovať autorov a podporiť ich v zlepšovaní.

- **Inšpekcia**. Vzhľadom na to, že inšpekcie sú najformálnejším typom revízie, riadia sa komplexným všeobecným procesom (pozri *kapitolu 3.2.2*). Hlavným cieľom je nachádzať maximálny počet anomálií. Ďalšími cieľmi sú hodnotenie kvality, budovanie dôvery v pracovný produkt, motivácia a podpora autorov v zlepšovaní. Metriky sú zhromažďované a používané za účelom zlepšovania celého SDLC, vrátane samotného procesu inšpekcie. Pri inšpekciách nemôže autor vystupovať ako vedúci revízie alebo zapisovateľ.

### 3.2.5 Faktory úspechu pri revízii

Existuje niekoľko faktorov kľúčových pre úspech procesu revízie:

- Sú definované jasné ciele a merateľné výstupné kritériá. Nie sú hodnotení účastníci revízie, ale revidovaný pracovný produkt.
- Je vybraný taký typ revízie, ktorý je vhodný na dosiahnutie daných cieľov, pre typ pracovného produktu, pre účastníkov revízie a pre potreby a kontext projektu.
- Revízie sú vykonávané po malých častiach, takže revidujúci nestrácajú koncentráciu počas revízie a/alebo počas revíznych schôdzok (ak sa konajú).
- Zainteresovaným stranám a autorom je poskytovaná spätná väzba z revízií tak, aby mohli zlepšovať produkt a svoje činnosti (pozri *kapitolu 3.2.1*).
- Účastníci majú dostatok času na prípravu.
- Manažment podporuje proces revízie.
- Revízie sú súčasťou firemnej kultúry s cieľom podporovať učenie a zlepšovanie procesov.
- Je poskytnuté vhodné odborné školenie pre všetkých účastníkov tak, aby vedeli, ako plniť svoju rolu v procese.
- Schôdze sú správne riadené.

# 4 Testovacia analýza a návrh testov – 390 minút

**Kľúčové slová**

akceptačné kritériá, analýza hraničných hodnôt, odhadovanie chýb, pokrytie, pokrytie príkazov, pokrytie vetiev, položka pokrytia, prieskumné testovanie, rozdelenie na triedy ekvivalencie, technika testovania, technika testovania bielej skrinky, technika testovania založená na skúsenostiach, technika testovania čiernej skrinky, testovací prístup založený na spolupráci, testovanie prechodov stavov, testovanie rozhodovacou tabuľkou, testovanie založené na kontrolných zoznamoch, vývoj riadený akceptačnými testami

**Študijné ciele**

### 4.1 Prehľad techník testovania

FL-4.1.1 (K2) Rozlíšiť medzi technikami testovania čiernej skrinky, bielej skrinky a technikami založenými na skúsenostiach.

### 4.2 Techniky testovania čiernej skrinky

FL-4.2.1 (K3) Použiť techniku rozdelenia na triedy ekvivalencie na odvodenie testovacích prípadov.

FL-4.2.2 (K3) Použiť techniku analýzy hraničných hodnôt na odvodenie testovacích prípadov.

FL-4.2.3 (K3) Použiť techniku testovania podľa rozhodovacej tabuľky na odvodenie testovacích prípadov.

FL-4.2.4 (K3) Použiť techniku testovania prechodov stavov na odvodenie testovacích prípadov.

### 4.3 Techniky testovania bielej skrinky

FL-4.3.1 (K2) Vysvetliť techniku testovania príkazov.

FL-4.3.2 (K2) Vysvetliť techniku testovania vetiev.

FL-4.3.3 (K2) Vysvetliť prínos testovania bielej skrinky.

### 4.4 Techniky testovania založené na skúsenostiach

FL-4.4.1 (K2) Vysvetliť techniku odhadovania chýb.

FL-4.4.2 (K2) Vysvetliť techniku prieskumného testovania.

FL-4.4.3 (K2) Vysvetliť techniku testovania založenú na kontrolných zoznamoch.

### 4.5 Prístupy k testovaniu založené na spolupráci

FL-4.5.1 (K2) Vysvetliť, ako písať používateľské scenáre v spolupráci s vývojármi a zástupcami biznisu.

FL-4.5.2 (K2) Kategorizovať rôzne možnosti písania akceptačných kritérií.

FL-4.5.3 (K3) Použiť vývoj riadený akceptačnými testami na odvodenie testovacích prípadov.

## 4.1 Prehľad techník testovania

Techniky testovania podporujú testerov pri testovacej analýze (v zmysle „čo testovať“) a pri návrhu testov (v zmysle „ako testovať“). Pomáhajú systematicky definovať relatívne malú, ale dostatočnú sadu testovacích prípadov. Techniky testovania tiež pomáhajú testerom definovať testovacie podmienky, identifikovať položky pokrytia a identifikovať testovacie dáta. Ďalšie informácie o technikách testovania a im zodpovedajúcich opatreniach možno nájsť v norme ISO/IEC/IEEE 29119-4 (*Softwarové a systémové inženýrství – Testování softwaru – Část 4: Techniky testování*, 2021) a v (Beizer, 1990), (Craig et al., 2002), (Copeland, 2004), (Vroon et al., 2006), (Jorgensen, 2014), (Ammann et al., 2016), (Forgacs et al., 2019).

V týchto učebných osnovách sú techniky testovania klasifikované ako čierna skrinka, biela skrinka a založené na skúsenostiach.

**Techniky testovania čiernej skrinky** (známej tiež ako techniky založené na špecifikáciách) sú založené na analýze špecifikovaného správania testovaného objektu odkazu na jeho vnútornú štruktúru. Testovacie prípady sú teda nezávislé od toho, ako je softvér implementovaný. Ak sa teda implementácia zmení, ale požadované správanie zostane rovnaké, testovacie prípady sú stále užitočné.

**Techniky testovania bielej skrinky** (známe tiež ako techniky založené na štruktúre) sú založené na analýze vnútornej štruktúry a spôsobe spracovania testovaného objektu. Keďže testovacie prípady závisia od toho, ako je softvér navrhnutý, možno ich vytvoriť až po návrhu alebo implementácii testovacieho objektu.

**Techniky testovania založené na skúsenostiach** efektívne využívajú znalosti a skúsenosti testerov pri návrhu a implementácii testovacích prípadov. Efektivita týchto techník silne závisí od zručnosti testerov. Techniky testovania založené na skúsenostiach môžu odhaliť chyby, ktoré môžu byť prehliadnuté pomocou techník testovania čiernej skrinky a bielej skrinky. Preto techniky testovania založené na skúsenostiach dopĺňajú techniky testovania čiernej a bielej skrinky.

## 4.2 Techniky testovania čiernej skrinky

Bežne používané techniky testovania čiernej skrinky sú popísané v nasledujúcich kapitolách:

- rozdelenie na triedy ekvivalencie,
- analýza hraničných hodnôt,
- testovanie rozhodovacou tabuľkou,
- testovanie prechodov stavov.

### 4.2.1 Rozdelenie na triedy ekvivalencie

Rozdelenie na triedy ekvivalencie (EP – equivalence partitioning) je technika testovania rozdeľujúca dáta do tried (označovaných ako triedy ekvivalencie) na základe očakávania, že všetky prvky danej triedy budú testovacím objektom spracované rovnakým spôsobom. Teória stojaca za touto technikou spočíva v tom, že pokiaľ testovací prípad detekuje defekt pre niektorú hodnotu z určitej triedy ekvivalencie, mal by tento testovací prípad odhaliť defekt aj pre akúkoľvek inú hodnotu tej istej triedy.

Triedy ekvivalencie je možné identifikovať pre ľubovoľný dátový element súvisiaci s testovaným objektom ako sú napr. vstupy, výstupy, konfiguračné položky, interné hodnoty, hodnoty súvisiace s časom alebo

parametrami rozhrania. Triedy môžu byť spojité alebo diskrétne, usporiadané alebo neusporiadané, konečné alebo nekonečné. Triedy sa nesmú prekrývať a musia ísť o neprázdne množiny.

Pre jednoduché testované objekty môže byť aplikácia tejto techniky jednoduchá, ale v praxi je často komplikované pochopiť, ako bude testovací objekt zaobchádzať s rôznymi hodnotami. Preto by sa rozdelenie do tried malo vykonávať opatrne.

Trieda ekvivalencie, ktorá obsahuje platné hodnoty, sa nazýva platná trieda ekvivalencie. Trieda ekvivalencie, ktorá obsahuje neplatné hodnoty, sa nazýva neplatná trieda ekvivalencie. Definície platných a neplatných hodnôt sa môžu v jednotlivých tímoch a organizáciách líšiť. Napríklad platné hodnoty sa môžu interpretovať ako hodnoty, ktoré by mal testovací objekt spracovať, alebo ako hodnoty, pre ktoré špecifikácia definuje ich spracovanie. Neplatné hodnoty sa môžu interpretovať ako hodnoty, ktoré by mal testovací objekt ignorovať alebo odmietnuť, alebo ako hodnoty, pre ktoré nie je v špecifikácii testovacieho objektu definované žiadne spracovanie.

V EP sú položkami pokrytia samotné triedy ekvivalencie. Na dosiahnutie 100% pokrytia musia testovacie prípady pokrývať všetky identifikované triedy ekvivalencie (vrátane neplatných), a to použitím minimálne jednej hodnoty z každej triedy. Pokrytie sa meria ako pomer počtu tried otestovaných aspoň jedným testovacím prípadom k celkovému počtu identifikovaných tried (obvykle vyjadrené v percentách).

Mnohé testovacie objekty obsahujú viacero množín tried ekvivalencie (napr. testované objekty s viacerými vstupnými parametrami). V tomto prípade môže jeden testovací prípad pokrývať triedy z rôznych množín tried. Najjednoduchším kritériom pokrytia v takýchto prípadoch je tzv. pokrytie každej voľby – pozri (Ammann et al., 2016). Toto kritérium vyžaduje, aby testovacie prípady pokryli každú triedu z každej množiny tried aspoň raz a zároveň neberie do úvahy kombinácie tried.

### 4.2.2 Analýza hraničných hodnôt

Analýza hraničných hodnôt (BVA – boundary value analysis) je technika založená na pokrytí okrajových hodnôt tried ekvivalencií. Preto ju možno použiť iba pre usporiadané (zoradené) triedy, kde minimálne a maximálne hodnoty každej triedy sú jej hraničné hodnoty. Pre túto techniku platí, že pokiaľ dva prvky patria do rovnakej triedy, všetky prvky medzi nimi musia tiež patriť do tejto triedy.

BVA sa zameriava na hraničné hodnoty tried, pretože vývojári sa pri týchto hraničných hodnotách častejšie dopúšťajú chýb. Typické defekty zistené technikou BVA sa nachádzajú v tých oblastiach, kde sú implementované (skutočné) hranice nesprávne presunuté na pozície nad alebo pod ich špecifikovanými (predpokladanými) hodnotami, prípadne kde sú úplne vynechané.

Tieto učebné osnovy sa zaoberajú dvoma variantmi: 2-hodnotová a 3-hodnotová BVA. Tieto dva varianty sa líšia v počte položiek pokrytia na danej hranici, potrebných na dosiahnutie 100% pokrytia.

V prípade 2-hodnotovej BVA (Craig et al., 2002), (Myers et al., 2011) existujú pre každú hranicu dve položky pokrytia: hraničná hodnota a jej najbližší sused patriaci do susednej triedy. Na dosiahnutie 100% pokrytia s týmto variantom musia byť testovacie prípady vykonané pre všetky položky pokrytia (t. j. pre všetky identifikované hraničné hodnoty). Pokrytie sa meria ako pomer počtu vykonaných hraničných hodnôt k celkovému počtu identifikovaných hraničných hodnôt (obvykle vyjadrené v percentách).

V trojhodnotovom BVA (Vroon et al., 2006), (O'Regan, 2019) sú pre každú hraničnú hodnotu tri položky pokrytia: táto hraničná hodnota a obe jej susedné hodnoty. Preto v 3-hodnotovej BVA niektoré z položiek pokrytia nemusia byť hraničnými hodnotami. Na dosiahnutie 100% pokrytia s týmto variantom musia byť testovacie prípady vykonané pre všetky položky pokrytia, t. j. pre identifikované hraničné hodnoty aj pre všetkých susedov. Pokrytie sa meria ako pomer počtu vykonaných hraničných hodnôt a ich susedných

hodnôt vydelený k celkovému počtu identifikovaných hraničných hodnôt a ich susedných hodnôt (obvykle vyjadrené v percentách).

3-hodnotová BVA je „prísnejšia“ ako 2-hodnotová BVA, s jej pomocou je možné odhaliť defekty, ktoré by mohli byť s použitím 2-hodnotové BVA prehliadnuté. Napríklad, ak je podmienka "IF ($x \leq 10$) ..." nesprávne implementovaná ako "IF (x = 10) ...", žiadny z testovacích prípadov vytvorených pomocou 2-hodnotovej BVA (x = 10, x = 11) nemôže túto chybnú implementáciu (a následný defekt) odhaliť. Testovací prípad x = 9 odvodený pomocou 3-hodnotovej BVA túto chybu pravdepodobne odhalí.

### 4.2.3 Testovanie rozhodovacou tabuľkou

Rozhodovacie tabuľky sa používajú na testovanie implementácie systémových požiadaviek, ktoré špecifikujú, ako rôzne kombinácie podmienok vedú k rôznym výsledkom. Predstavujú efektívny spôsob zaznamenávania zložitej logiky ako sú napríklad biznisové pravidlá.

Pri vytváraní rozhodovacích tabuliek sa definujú podmienky a výsledné akcie systému, ktoré tvoria dve skupiny riadkov tabuľky. Každý stĺpec zodpovedá jednému pravidlu rozhodovania, ktoré definuje jedinečnú kombináciu podmienok vedúcich k vykonaniu akcií spojených s týmto pravidlom. V rozhodovacích tabuľkách s obmedzeným počtom vstupov sú všetky hodnoty podmienok a akcií (s výnimkou irelevantných alebo neuskutočniteľných hodnôt, pozri nižšie) zobrazujú ako logické hodnoty (pravda alebo nepravda).

Alternatívne, v rozhodovacích tabuľkách s rozšíreným počtom vstupov môžu niektoré (alebo všetky) podmienky a akcie tiež nadobúdať viac hodnôt (napr. rozsahy hodnôt, triedy ekvivalencie, diskrétne hodnoty).

Zápis podmienok je nasledovný:

- „T“ (true) – podmienka je splnená,
- „F“ (false) – podmienka nie je splnená,
- „–“ znamená, že hodnota podmienky je pre výsledok akcie (výstup) irelevantná,
- „N/A“ – podmienka je pre dané pravidlo neuskutočniteľná.

Zápis akcií (výstupov) je nasledujúci:

- „X“ – akcia by mala nastať,
- „“ (prázdna hodnota) – akcia by nemala nastať.

Môžu sa použiť aj iné zápisy.

Úplná rozhodovacia tabuľka obsahuje taký počet stĺpcov, aby došlo k pokrytiu každej kombinácie podmienok. Tabuľku je možné zjednodušiť odstránením stĺpcov obsahujúcich neuskutočniteľnú kombináciu podmienok alebo tiež zlúčením stĺpcov, v ktorých niektoré podmienky neovplyvňujú výsledok, do jedného stĺpca. Popis týchto algoritmov je ale nad rámec týchto osnov.

Pri rozhodovacích tabuľkách sú položkami pokrytia stĺpce obsahujúce uskutočniteľnú kombináciu podmienok. Na dosiahnutie 100% pokrytia touto technikou musia testovacie prípady pokryť všetky takéto stĺpce. Pokrytie sa meria ako pomer počtu pokrytých stĺpcov k celkovému počtu uskutočniteľných stĺpcov (zvyčajne vyjadrené v percentách).

Silnou stránkou testovania rozhodovacou tabuľkou je, že poskytuje systematický prístup na identifikáciu všetkých kombinácií podmienok, z ktorých by niektoré mohli byť inak prehliadnuté. Pomáha tiež nájsť

prípadné medzery alebo rozpory v požiadavkách.

Ak je podmienok veľa, môže byť preverovanie všetkých rozhodovacích pravidiel časovo náročné, pretože počet pravidiel rastie exponenciálne s počtom podmienok. V takom prípade sa na zníženie počtu pravidiel, ktoré je potrebné vykonať, môže použiť minimalizovaná rozhodovacia tabuľka alebo prístup založený na riziku.

### 4.2.4 Testovanie prechodov stavov

Diagram prechodov stavov modeluje správanie systému zobrazením jeho možných stavov a platných prechodov medzi nimi. Prechod je iniciovaný výskytom udalosti a môže byť doplnený o podmienku prechodu, ktorej splnenie prechod podmieňuje. Predpokladá sa, že prechody sú vykonávané okamžite a niekedy môžu viesť k tomu, že softvér vykoná určitú akciu. Bežná syntax označovania prechodov je „udalosť [podmienka prechodu] / akcie“. Pokiaľ podmienky prechodov alebo akcie neexistujú (príp. sú pre testerov irelevantné), môžu byť vynechané.

K diagramu prechodov stavov je ekvivalentná aj tabuľka prechodov stavov. Jej riadky predstavujú stavy a jej stĺpce udalosti (doplnené o podmienky prechodov ak existujú). Jednotlivé položky tabuľky (bunky) predstavujú prechody a obsahujú cieľový stav spolu s podmienkami prechodov a výslednými akciami (ak sú definované). Na rozdiel od diagramu prechodov stavov, zobrazuje tabuľka prechodov stavov aj neplatné prechody, ktoré sú reprezentované prázdnymi bunkami.

Testovací prípad odvodený z diagramu prechodov stavov alebo z tabuľky prechodov stavov je obvykle znázornený ako séria udalostí vedúcich k postupnosti zmien stavov (a akcií, ak sú definované). Jeden testovací prípad môže (a obvykle bude) pokrývať niekoľko prechodov medzi stavmi.

Existuje mnoho kritérií pokrytia pre testovanie prechodov stavov. Tieto učebné osnovy popisujú tri z nich.

Pri **pokrytí všetkých stavov** sú položkami pokrytia príslušné stavy. Aby bolo dosiahnuté 100% pokrytie všetkých stavov, musia testovacie prípady zabezpečiť pokrytie všetkých stavov. Pokrytie sa meria ako pomer počtu pokrytých stavov k celkovému počtu stavov (obvykle vyjadrené v percentách).

Pri **pokrytí platných prechodov** (nazývanom tiež pokrytie 0-switch) sú položkami pokrytia jednotlivé platné prechody. Na dosiahnutie 100% pokrytia platných prechodov musia testovacie prípady pokryť všetky platné prechody. Pokrytie sa meria ako pomer počtu pokrytých platných prechodov k celkovému počtu platných prechodov (obvykle vyjadrené v percentách).

Pri **pokrytí všetkých prechodov** sú položkami pokrytia všetky prechody v tabuľke prechodov stavov. Na dosiahnutie 100% pokrytia všetkých prechodov musia testovacie prípady preveriť všetky platné prechody a pokúsiť sa preveriť prechody neplatné (tzn. pokryť ako platné, tak neplatné prechody). Testovanie iba jedného neplatného prechodu v jednom testovacom prípade pomáha vyhnúť sa maskovaniu chýb, t. j. situácii, keď jedna chyba bráni odhaleniu inej. Pokrytie sa meria ako pomer počtu platných a neplatných prechodov, ktoré boli alebo mali byť pokryté vykonanými testovacími prípadmi, k celkovému počtu všetkých platných a neplatných prechodov (obvykle vyjadrené v percentách).

Kritérium pokrytia všetkých stavov je „slabšie“ ako kritérium pokrytia platných prechodov, pretože môže byť obvykle dosiahnuté bez vykonania všetkých prechodov. Najpoužívanejším kritériom pokrytia je ale pokrytie platných prechodov, ktoré by tiež malo byť minimálnou požiadavkou pre bezpečnostne kritický softvér.

Platí, že dosiahnutie plného pokrytia prechodov garantuje plné pokrytie všetkých stavov a všetkých platných prechodov.

## 4.3 Techniky testovania bielej skrinky

Táto kapitola sa zameriava na dve najznámejšie techniky testovania bielej skrinky, s pomocou ktorých je možné otestovať kód:

- testovanie príkazov,
- testovanie vetiev.

Existujú prísnejšie techniky, ktoré sa používajú v niektorých prostrediach kritických z hľadiska bezpečnosti (safety-critical), kritických z hľadiska použitia (mission-critical) alebo vysokej integrity (high-integrity) na dosiahnutie dôkladnejšieho pokrytia kódu. Techniky testovania bielej skrinky nesúvisia výhradne s úrovňou testovania komponentov. Existujú aj techniky testovania bielej skrinky používané vo vyšších úrovniach testovania (napr. pri API testovaní) alebo využívajú pokrytie, ktoré nesúvisí s kódom (napr. pokrytie neurónov pri testovaní neurónových sietí). Tieto techniky nie sú v týchto učebných osnovách rozoberané.

### 4.3.1 Testovanie a pokrytie príkazov

Pri testovaní príkazov sú položkami pokrytia spustiteľné príkazy. Cieľom je navrhovať také testovacie prípady, ktoré pokryjú príkazy v kóde, kým sa nedosiahne akceptovateľná úroveň pokrytia. Pokrytie sa meria ako pomer počtu spustiteľných príkazov pokrytých testami k celkovému počtu spustiteľných príkazov v kóde (obvykle vyjadrené v percentách).

Ak sa dosiahne 100% pokrytie príkazov, zabezpečí sa, že všetky vykonateľné príkazy v kóde boli vykonané aspoň raz. To znamená, že bude spustený každý príkaz, ktorý môže spôsobiť zlyhanie indikujúce výskyt defektu. Opäť ale platí, že otestovanie príkazu pomocou testovacieho prípadu nemusí vždy odhaliť defekt, technika napríklad nemusí odhaliť defekty súvisiace s dátami (napr. delenie nulou, ktoré zlyhá iba pri nulovej hodnote deliteľa). Takisto 100% pokrytie príkazov nezaručuje, že všetka rozhodovacia logika bola otestovaná, pretože nemusí dôjsť k spusteniu všetkých vetiev v kóde (pozri *kapitolu 4.3.2*).

### 4.3.2 Testovanie a pokrytie vetiev

Vetva je prenos riadenia medzi dvoma uzlami v grafe riadiaceho toku zobrazujúceho možné sekvencie, v ktorých sú príkazy zdrojového kódu v testovacom objekte vykonávané. Každý prenos riadenia môže byť buď nepodmienený (t. j. lineárny kód) alebo podmienený (t. j. výsledok rozhodnutia).

Pri testovaní vetiev sú položkami pokrytia vetvy a cieľom je navrhovať testovacie prípady tak, aby došlo k pokrytiu vetiev v kóde, kým sa nedosiahne akceptovateľná úroveň pokrytia. Pokrytie sa meria ako pomer počtu vetiev pokrytých testami k celkovému počtu vetiev (obvykle vyjadrené v percentách).

Pri dosiahnutí 100% pokrytia vetiev sú všetky nepodmienené a podmienené vetvy v kóde pokryté testovacími prípadmi. Podmienené vetvy zvyčajne zodpovedajú pravdivému alebo nepravdivému výsledku z podmienky (IF / THEN / ELSE), výsledku z viaccestnej podmienky (SWITCH / CASE) alebo rozhodnutiu, či ukončiť alebo pokračovať v slučke. Vykonávanie vetvy pomocou testovacieho prípadu však neodhalí chyby vo všetkých prípadoch. Nemusí napríklad odhaliť chyby vyžadujúce vykonanie konkrétnej cesty v kóde.

Pokrytie vetiev automaticky zaručuje pokrytie príkazov. To znamená, že akákoľvek sada testovacích prípadov, ktorá dosiahne 100% pokrytie vetiev, dosiahne tiež 100% pokrytie príkazov (ale nie naopak).

### 4.3.3 Význam testovania bielej skrinky

Silnou stránkou techník bielej skrinky je, že pri testovaní je zohľadnená samotná implementácia softvéru, čo uľahčuje odhalenie defektov v prípadoch, keď je jeho špecifikácia vágna, zastaraná alebo neúplná. Zodpovedajúcou slabinou je, že ak softvér neimplementuje jednu alebo viacero požiadaviek, testovanie bielych skriniek nemusí odhaliť výsledné chyby z opomenutia (Watson et al., 1996).

Techniky bielej skrinky môžu byť použité pri statickom testovaní (napr. počas tzv. behov kódu „nanečisto“, dry-runs). Sú tiež vhodné pri revízii kódu, ktorý ešte nie je pripravený na spustenie (Hetzel, 1988) rovnako ako pri revízii pseudokódu a inej vysokoúrovňovej alebo zhora nadol orientovanej logiky, ktorú možno modelovať pomocou grafu riadiaceho toku.

Samotné testovanie pomocou techník čiernej skrinky neposkytuje informácie o miere skutočného pokrytia kódu. Naopak výsledky testovania s využitím techník bielej skrinky poskytujú objektívne meranie pokrytia a poskytujú tiež nevyhnutné informácie umožňujúce vytváranie ďalších testov s cieľom zvyšovania tohto pokrytia a tým aj následné zvyšovanie dôvery v kód.

## 4.4 Techniky testovania založené na skúsenostiach

Bežne používané techniky testovania založené na skúsenostiach popísané v nasledujúcich kapitolách sú:

- odhadovanie chýb,
- prieskumné testovanie,
- testovanie založené na kontrolnom zozname.

### 4.4.1 Odhadovanie chýb

Odhadovanie chýb je technika používaná na predvídanie výskytu chýb, defektov a zlyhaní, ktorá je založená na znalostiach testerov, vrátane:

- ako fungovala aplikácia v minulosti
- typy chýb, ktorých sa vývojári dopúšťajú, a typy chýb, ktoré z nich vyplývajú
- typy zlyhaní, ktoré sa vyskytli v iných podobných aplikáciách

Všeobecne platí, že chyby, defekty a zlyhania môžu súvisieť so vstupom (napr. nebol akceptovaný správny vstup, nesprávne alebo chýbajúce parametre), výstupom (napr. nesprávny formát alebo nesprávny výsledok), logikou (napr. chýbajúce prípady, nesprávny operátor), výpočtom (napr. nesprávny operand, nesprávny výpočet), rozhraniami (napr. nesúlad parametrov, nekompatibilné typy) alebo dátami (napr. nesprávna inicializácia, nesprávny typ).

Známym metodickým prístupom k implementácii odhadovania chýb je útok na vady (fault attack). Táto technika vyžaduje, aby tester vytvoril alebo získal zoznam možných chýb, defektov a zlyhaní a navrhol testy, ktoré identifikujú chyby, odhalia defekty alebo spôsobia zlyhania. Tieto zoznamy je možné zostaviť na základe skúseností, údajov o defektoch a zlyhaniach alebo na základe všeobecných znalostí o tom, prečo softvér zlyháva.

Viac informácií o odhadovaní chýb a útokoch na vady nájdete v (Whittaker, 2003), (Whittaker, 2009), (Andrews et al., 2006).

### 4.4.2 Prieskumné testovanie

Pri prieskumnom testovaní sú testy súčasne navrhované, vykonávané a vyhodnocované, zatiaľ čo testeri získavajú znalosti o testovanom objekte. Testovanie sa (okrem iného) používa k získaniu ďalších informácií o testovanom objekte, na jeho hlbšie preskúmanie pomocou cielených testov a na vytvorenie testov pre netestované oblasti.

Prieskumné testovanie sa niekedy vykonáva pomocou tzv. testovania v sedeniach, ktoré pomáhajú štruktúrovať testovanie. Testovanie sa vykonáva v rámci definovaného časového rámca. Tester používa testovaciu listinu obsahujúcu ciele testovania, ktorou sa testovanie riadi. Po testovacej relácii obvykle nasleduje tzv. debriefing s diskusiou medzi testerom a zainteresovanými stranami, ktoré sa zaujímajú o výsledky testovacieho sedenia.

Všeobecnými testovacími podmienkami môžu byť pri prieskumnom testovaní ciele testovania. Jednotlivé položky pokrytia sú identifikované a preverované počas testovacieho sedenia a testeri môžu používať dokumenty pre testovacie sedenie (test session sheets) pre zdokumentovanie vykonávaných krokov a nových zistení.

Prieskumné testovanie je najviac užitočné v prípadoch, keď je špecifikácia nedostatočná alebo úplne chýba, taktiež je vhodnou metódou v prípade výrazného časového tlaku na testovanie. Je tiež užitočné ako doplnok iných (formálnejších) testovacích techník. Všeobecne bude efektívnejšie v prípadoch, keď sú testeri skúsení, majú znalosti z danej domény a vysokú úroveň základných zručností ako sú analytické schopnosti, zvedavosť a kreativita (pozri *kapitolu 1.5.1*).

Prieskumné testovanie môže využívať aj iné testovacie techniky (napr. rozdelenie tried ekvivalencie). Viac informácií o prieskumnom testovaní, pozri (Kaner; Falk et al., 1999), (Whittaker, 2003) a (Hendrickson, 2013).

### 4.4.3 Testovanie založené na kontrolných zoznamoch

Pri testovaní založenom na kontrolných zoznamoch tester navrhuje, implementuje a vykonáva testy, ktoré pokrývajú testovacie podmienky z kontrolného zoznamu. Kontrolné zoznamy môžu byť zostavené na základe skúseností, poznatkov o tom, čo je dôležité pre používateľa, alebo na základe pochopenia toho, prečo a ako softvér zlyháva. Kontrolné zoznamy by nemali obsahovať položky, ktoré sa dajú skontrolovať automaticky, položky, ktoré sú vhodnejšie ako vstupné alebo výstupné kritériá, alebo položky, ktoré sú príliš všeobecné (Brykczynski, 1999).

Položky kontrolného zoznamu sú často formulované vo forme otázky. Každú položku by malo byť možné priamo skontrolovať, a to oddelene (nezávisle od ostatných položiek). Tieto položky sa môžu týkať požiadaviek, vlastností grafického rozhrania (UI), kvalitatívnych charakteristík alebo iných foriem testovacích podmienok. Kontrolné zoznamy môžu byť taktiež vytvorené ako podpora rôznych typov testovania vrátane funkcionálneho a nefunkcionálneho testovania (napr. 10 princípov testovania použiteľnosti, pozri (Nielsen, 1994)).

Niektoré položky kontrolného zoznamu môžu byť časom menej efektívne, pretože vývojári sa postupne naučia vyhýbať rovnakým chybám. Kontrolné zoznamy by preto mali byť pravidelne aktualizované na základe analýzy defektov, typicky je možné do nich pridávať nové položky reflektujúce výskyt novo zistených defektov s vysokou závažnosťou. Treba však dbať na to, aby sa kontrolný zoznam nestal príliš dlhým (Gawande, 2011).

Pri absencii podrobných testovacích prípadov môže testovanie založené na kontrolných zoznamoch poskytnúť vodítko a určitý stupeň konzistencie pre testovanie. Keďže ide o všeobecné zoznamy (high-level

sheets), je pravdepodobné, že sa v testovaní môže objaviť variabilita, čo môže mať za následok väčšie pokrytie, ale menšiu opakovateľnosť testovania.

## 4.5 Prístupy k testovaniu založené na spolupráci

Každá z vyššie uvedených techník (pozri *kapitolu 4.2*, *kapitolu 4.3*, *kapitolu 4.4*) má svoj špecifický cieľ vo vzťahu k identifikácii defektov. Na druhej strane prístupy založené na spolupráci sa zameriavajú aj na predchádzanie vzniku chýb prostredníctvom spolupráce a komunikácie.

### 4.5.1 Spoločné písanie používateľských scenárov

Používateľský scenár (user story) predstavuje funkcionalitu (feature), ktorá má pre používateľa alebo pre budúceho majiteľa softvéru/systému určitú hodnotu. Používateľské scenáre obsahujú tri dôležité aspekty (pozri (Jeffries et al., 2001)), ktoré sa spoločne nazývajú "3 C":

- **karta** (card) – médium popisujúce používateľský scenár (napr. farebné štítky alebo záznam na elektronickej tabuli v online systéme),
- **konverzácia** (conversation) – vysvetlenie, ako sa bude softvér používať (môže byť dokumentovaná alebo byť iba verbálna),
- **potvrdenie** (confirmation) – vo forme akceptačného kritéria (pozri *kapitolu 4.5.2*).

Najbežnejším formátom používateľského scenára je formula „Ako [ROLA] chcem, aby [CIEĽ, ktorý má byť splnený], pretože [výsledná biznisová HODNOTA pre rolu]" doplnená akceptačnými kritériami.

Spoločné autorstvo používateľského scenára môže využívať techniky ako je brainstorming alebo tvorba myšlienkových máp (mind mapping). Spolupráca umožňuje tímu získať spoločnú víziu toho, čo by sa malo dodať, zohľadnením troch perspektív: (biznis, vývoj, testovanie).

Správne používateľské scenáre by mali byť tzv. INVEST: nezávislé (Independent), schodné (Negotiable), hodnotné (Valuable), odhadnuteľné (Estimable), malé (Small) a testovateľné (Testable). Ak zainteresovaná strana nevie, ako testovať používateľský scenár, môže to znamenať, že používateľský scenár nie je dostatočne jasný alebo že nevyjadruje niečo, čo je pre ňu hodnotné, alebo že zainteresovaná strana jednoducho potrebuje pomoc pri testovaní (Wake, 2003).

### 4.5.2 Akceptačné kritériá

Akceptačné kritériá pre používateľský scenár sú podmienky, ktoré musí jeho implementácia spĺňať, aby bola akceptovaná zainteresovanými stranami. Z tohto hľadiska je možné na ne pozerať ako na testovacie podmienky, ktoré by mali testy preveriť. Akceptačné kritériá sú zvyčajne výsledkom konverzácie (pozri *kapitolu 4.5.1*).

Akceptačné kritériá sa používajú na:

- definovanie rozsahu používateľského scenára,
- dosiahnutie zhody medzi zainteresovanými stranami,
- popis pozitívnych aj negatívnych scenárov,
- základnú definíciu akceptačného testovania používateľského scenára (pozri *kapitolu 4.5.3*),

- presnejšie plánovanie a odhadovanie.

Existuje niekoľko spôsobov, ako napísať akceptačné kritériá pre používateľský scenár. Dva najbežnejšie formáty sú:

- orientovaný na scenáre (napr. formát Given/When/Then používaný v BDD, pozri *kapitolu 2.1.3*),
- orientovaný na pravidlá (napr. verifikačný zoznam s odrážkami alebo tabuľka s mapovaním vstupov a výstupov).

Väčšinu akceptačných kritérií možno zdokumentovať v jednom z týchto dvoch formátov. Tím však môže použiť aj iný, vlastný formát, pokiaľ sú akceptačné kritériá dobre definované a jednoznačné.

### 4.5.3 Vývoj riadený akceptačnými testami (ATDD)

ATDD je jeden z prístupov iniciovaných testami (test-first) (pozri *kapitolu 2.1.3*), kedy sú testovacie prípady vytvorené pred vlastnou implementáciou používateľského scenára členmi tímu s rôznymi perspektívami pohľadu, ako napr. zákazníci, vývojári a testeri (Adzic, 2009). Testovacie prípady sa môžu vykonávať manuálne alebo automatizovane.

Prvým krokom je obvykle schôdzka nad špecifikáciami, kde členovia tímu analyzujú, diskutujú a dokumentujú používateľské scenáre a (ak ešte nie sú definované) ich akceptačné kritériá. Počas tohto procesu sú obvykle vyriešené neúplnosti, nejednoznačnosti alebo defekty v používateľskom scenári.

Ďalším krokom je vytvorenie testovacích prípadov. Môže to urobiť tím ako celok alebo tester individuálne. Testovacie prípady sú založené na akceptačných kritériách a možno ich považovať za príklady toho, ako by mal softvér pracovať, čo pomôže tímu správne implementovať používateľský scenár. Keďže príklady a testy sú rovnaké, tieto pojmy sú zameniteľné. Prvé testovacie prípady sú zvyčajne pozitívne, potvrdzujú správne chovanie bez výnimiek alebo chybových stavov a obsahujú postupnosť vykonávaných činností v prípadoch, keď všetko prebieha podľa očakávania. Po dokončení pozitívnych testovacích prípadov by mal tím vykonať negatívne testovanie a nakoniec pokryť nefunkcionálne charakteristiky kvality (napr. výkonnostná efektivita alebo použiteľnosť). Pri návrhu testov je možné použiť techniky testovania popísané v kapitolách *4.2*, *4.3*, *4.4*.

Testovacie prípady by mali byť vyjadrené spôsobom, ktorý je pre zainteresované strany zrozumiteľný. Obvykle obsahujú vety v prirodzenom jazyku zahŕňajúce nevyhnutné vstupné podmienky (pre-conditions – ak existujú), vstupy a výstupné podmienky (post-conditions). Musia pokrývať všetky charakteristiky používateľského scenára a nemali by presahovať jeho rámec. Žiadne dva testovacie prípady by nemali popisovať rovnaké charakteristiky používateľského scenára.

Akceptačné kritériá môžu podrobne popisovať niektoré aspekty popísané v používateľskom scenári. Ak sú zachytené vo formáte podporovanom nejakým frameworkom na automatizáciu testov, vývojári môžu automatizovať testovacie prípady písaním podporného kódu pri implementácii funkcionality opísanej v používateľskom scenári. V takom prípade sa akceptačné testy vlastne stanú vykonateľnými požiadavkami.

# 5 Manažment testovania – 335 minút

**Kľúčové slová**

analýza rizík, identifikácia rizík, kvadranty testovania, manažment defektov, manažment rizík, monitorovanie rizík, monitorovanie testovania, ohodnotenie rizík, plán testovania, plánovanie testovania, produktové riziko, projektové riziko, prístup k testovaniu, report o defekte, riadenie rizík, riadenie testovania, riziko, správa o dokončení testovania, správa o postupe testovania, testovacia pyramída, testovanie založené na rizikách, úroveň rizika, vstupné kritériá, výstupné kritériá, zmierňovanie rizík

**Študijné ciele:**

### 5.1 Plánovanie testovania

FL-5.1.1 (K2) Ilustrovať na príkladoch účel a obsah plánu testovania.

FL-5.1.2 (K1) Určiť pridanú hodnotu prítomnosti testerov pre plánovanie iterácií a vydaní.

FL-5.1.3 (K2) Porovnať a uviesť odlišnosti vstupných a výstupných kritérií.

FL-5.1.4 (K3) Použiť techniky odhadovania na výpočet úsilia potrebného na testovanie.

FL-5.1.5 (K3) Použiť prioritizáciu testovacích prípadov.

FL-5.1.6 (K1) Zapamätať si koncept testovacej pyramídy.

FL-5.1.7 (K2) Zhrnúť kvadranty testovania a uvedomiť si ich vzťah k úrovniam a typom testovania.

### 5.2 Manažment rizík

FL-5.2.1 (K1) Identifikovať úroveň rizika pomocou pravdepodobnosti a vplyvu rizika.

FL-5.2.2 (K2) Rozlišovať medzi projektovými a produktovými rizikami.

FL-5.2.3 (K2) Vysvetliť, ako môže analýza produktových rizík ovplyvniť hĺbku a rozsah testovania.

FL-5.2.4 (K2) Vysvetliť, aké opatrenia možno prijať v reakcii na analyzované produktové riziká.

### 5.3 Monitorovanie, riadenie a dokončenie testovania

FL-5.3.1 (K1) Zapamätať si metriky používané pri testovaní.

FL-5.3.2 (K2) Zhrnúť účel a obsah správ z testovania a uvedomiť si, pre koho sú určené.

FL-5.3.3 (K2) Ilustrovať na príkladoch komunikáciu stavu testovania.

### 5.4 Konfiguračný manažment

FL-5.4.1 (K2) Zhrnúť, ako konfiguračný manažment prispieva k testovaniu.

### 5.5 Manažment defektov

FL-5.5.1 (K3) Pripraviť report o defekte.

# 5.1 Plánovanie testovania

## 5.1.1 Účel a obsah plánu testovania

Plán testovania popisuje ciele, zdroje a procesy testovania v rámci projektu. Plán testovania:

- dokumentuje spôsob dosiahnutia cieľov testovania a harmonogram testovania,
- pomáha zabezpečiť, aby vykonávané testovacie činnosti spĺňali stanovené kritériá,
- slúži ako spôsob komunikácie s členmi tímu a ďalšími zainteresovanými stranami,
- preukazuje, že testovanie je v súlade s existujúcou politikou a stratégiou testovania (prípadne vysvetľuje, prečo sa od nich testovanie odchyľuje).

Plánovanie testovania je vodítkom pri uvažovaní testerov a núti ich zamyslieť sa nad budúcimi výzvami súvisiacimi s rizikami, harmonogramami, ľuďmi, nástrojmi, nákladmi, potrebným úsilím, atď. Proces prípravy plánu testovania je užitočným nástrojom na analýzu úsilia potrebného na dosiahnutie cieľov testovania stanovených v rámci projektu.

Typický plán testovania obsahuje:

- kontext testovania (napr. rozsah, ciele testovania, obmedzenia, testovacia báza),
- predpoklady a obmedzenia testovacieho projektu,
- zainteresované strany (napr. roly, zodpovednosti, relevantnosť pre testovanie, potreby náboru a školenia),
- plán komunikácie (napr. spôsoby a frekvenciu komunikácie, šablóny pre dokumentáciu),
- register rizík (napr. produktové a projektové riziká),
- prístup k testovaniu (napr. úrovne testovania, typy testovania, techniky testovania, výstupy z testovania, vstupné a výstupné kritériá, nezávislosť testovania, požadované metriky, požiadavky na testovacie dáta, požiadavky na testovacie prostredie, odchýlky od politiky a stratégie testovania),
- rozpočet a harmonogram.

Ďalšie podrobnosti o pláne testovania a jeho obsahu je možné nájsť v norme ISO/IEC/IEEE 29119-3 (*Softwarové a systémové inženýrství – Testování softwaru – Část 3: Dokumentace testování* , 2021).

## 5.1.2 Prínos testerov pri plánovaní iterácií a vydaní

V iteratívnych SDLC sa obvykle vyskytujú dva druhy plánovania: plánovanie vydania a plánovanie iterácie.

**Plánovanie vydania** pozerá vpred smerom k vydaniu produktu, definuje a upravuje produktový backlog a môže zahŕňať rozpracovanie väčších používateľských scenárov do sady menších. Zároveň slúži ako základ pre definíciu prístupu k testovaniu a plánu testovania naprieč všetkými iteráciami. Testeri zapojení do plánovania vydania sa podieľajú na špecifikácii testovateľných používateľských scenárov a akceptačných kritérií (pozri *kapitolu 4.5*), podieľajú sa na analýzach projektových a produktových rizík (pozri *kapitolu 5.2*), odhadujú potrebné úsilie na testovanie používateľských scenárov (pozri *kapitolu 5.1.4*), stanovujú prístup k testovaniu a plánujú testovanie súvisiace s vydaním.

**Plánovanie iterácie** pozerá smerom ku koncu jednej iterácie a pracuje s backlogom iterácie. Testeri zapojení do plánovania iterácií sa podieľajú na podrobnej analýze rizík používateľských scenárov,

stanovujú ich testovateľnosť, podieľajú sa na ich rozklade do úloh (najmä pre testovacie činnosti), odhadujú potrebné úsilie na testovanie pre všetky testovacie činnosti a prispievajú k identifikácii a spresňovaniu funkcionálnych a nefunkcionálnych aspektov testovaného objektu.

### 5.1.3 Vstupné kritériá a výstupné kritériá

Vstupné kritériá určujú predpoklady pre realizáciu danej činnosti. Pokiaľ nie sú splnené vstupné kritériá, bude vykonanie testovacej činnosti pravdepodobne ťažšie, časovo náročnejšie, nákladnejšie a viac rizikové. Medzi typické vstupné kritériá patrí dostupnosť zdrojov (napr. ľudia, nástroje, prostredie, testovacie dáta, rozpočet, čas), dostupnosť testvéru (napr. testovacia báza, testovateľné požiadavky, používateľské scenáre, testovacie prípady) a počiatočná úroveň kvality testovaného objektu (napr. úspešnosť všetkých smoke testov).

Výstupné kritériá určujú podmienky, ktoré sa musia dosiahnuť, aby bolo možné vyhlásiť činnosť za dokončenú. Medzi typické výstupné kritériá patrí miera dôkladnosti (napr. dosiahnutá úroveň pokrytia, počet nevyriešených defektov, hustota defektov, počet neúspešných testovacích prípadov) a kritériá dokončenia (napr. vykonanie plánovaných testov, vykonanie statického testovania, zaznamenanie všetkých zistených defektov, automatizácia všetkých regresných testov).

Za platné výstupné kritériá možno tiež považovať vyčerpanie času alebo finančných prostriedkov. Za týchto okolností, je možné ukončiť testovanie aj v prípade nesplnenia ostatných výstupných kritérií, ak zainteresované strany preskúmali a akceptovali riziko nasadenia produktu do produkcie bez ďalšieho testovania.

V agilnom vývoji softvéru sa výstupné kritériá často nazývajú definícia hotového (Definition of Done - DoD) a určujú projektovú objektívnu metriku pre položky pripravené na vydanie. Vstupné kritériá, ktoré musí používateľský scenár spĺňať, aby sa mohli začať vývojové a/alebo testovacie činnosti, sa nazývajú definícia pripravenosti (Definition of Ready).

Vstupné a výstupné kritériá by mali byť definované pre každú úroveň testov a môžu sa líšiť v závislosti od stanovených cieľov testovania.

### 5.1.4 Techniky na odhadovanie

Odhad potrebného úsilia na testovanie reprezentuje očakávané množstvo práce, ktoré bude potrebné na dosiahnutie cieľov testovania v projekte. Je dôležité zainteresovaným stranám objasniť, že odhad vychádza z momentálnych predpokladov a vždy v sebe nesie aj určitú mieru chybovosti, pričom platí, že odhad pre malé úlohy je obvykle presnejší ako pre veľké úlohy. Preto je pri odhadovaní vhodné rozložiť rozsiahlu úlohu na sadu menších a tie následne odhadnúť. Tieto učebné osnovy popisujú štyri techniky odhadovania.

**Odhad na základe pomerov**. Pri tejto technike založenej na metrikách sa zhromažďujú údaje z predchádzajúcich projektov v rámci organizácie, čo umožňuje odvodiť „štandardizované“ pomerové metriky (vzorce) pre podobné projekty. Takéto metriky odvodené z projektov v rámci organizácie (napr. prevzaté z historických údajov) sú vo všeobecnosti najlepším zdrojom, ktorý možno v procese odhadovania použiť na odhad potrebného úsilia na testovanie nového projektu. Ak bola napríklad v predchádzajúcom projekte úsilie vynaložené na vývoj a úsilie vynaložené na testovania v pomere 3:2 a v súčasnom projekte sa očakáva, že úsilie potrebné na vývoj bude predstavovať 600 človeko-dní, dá sa úsilie potrebné na testovania odhadnúť na 400 človeko-dní.

**Extrapolácia**. Pri tejto technike založenej na metrikách sa v aktuálnom projekte vykonáva čo najskôr

meranie s cieľom zhromažďovať dáta. S dostatočným počtom pozorovaní (dát) možno potrebné zostávajúce úsilie aproximovať extrapoláciou týchto dát (obvykle použitím matematického modelu). Táto metóda je veľmi vhodná pre iteratívne modely SDLC, pretože tím môže napríklad extrapolovať úsilie potrebné na testovanie v nadchádzajúcej iterácii ako priemerne vynaložené úsilie z posledných troch iterácií.

**Wideband Delphi**. Pri tejto iteratívnej technike založenej na expertoch vykonávajú experti odhady založené na skúsenostiach. Každý taký expert (odborník) samostatne odhadne potrebné úsilie. Výsledky odhadov sú zhromaždené a pokiaľ sa objavia odchýlky mimo rozsahu dohodnutých hraníc, diskutujú experti o svojich aktuálnych odhadoch. Každý je potom požiadaný, aby na základe tejto spätnej väzby vykonal nový odhad, a to opäť samostatne. Tento proces sa opakuje, kým sa nedosiahne zhoda. Variantom Wideband Delphi, ktorá sa bežne používa pri agilnom vývoji softvéru je tzv. plánovací poker. Pri ňom sa odhady obvykle tvoria pomocou kariet s číslami, ktoré predstavujú mieru potrebného úsilia.

**Trojbodový odhad**. Pri tejto technike založenej na expertoch vykonávajú experti tri odhady: najoptimistickejší odhad (a), najpravdepodobnejší odhad (m) a najpesimistickejší odhad (b). Výsledný odhad (E) je ich vážený aritmetický priemer. V najpoužívanejšej verzii tejto techniky sa odhad vypočíta ako E = (a + 4 * m + b) / 6. Výhodou tejto techniky je, že umožňuje expertom vypočítať chybu merania, obvykle vo forme smerodajnej odchýlky: SD = (b - a) / 6. Ak sú odhady (v človeko-hodinách) napr.: a = 6, m = 9 a b = 18, potom je výsledný odhad 10 ± 2 človeko-hodín (t. j. 8 - 12 človeko-hodín), pretože E = (6 + 4 * 9 + 18) / 6 = 10 a SD = (18 - 6) / 6 = 2.

Viac informácií o týchto a mnohých ďalších technikách odhadovania úsilia potrebného na testovanie možno nájsť v (Kan, 2003), (Vroon et al., 2006) a (Westfall, 2016).

### 5.1.5 Prioritizácia testovacích prípadov

Akonáhle sú testovacie prípady a testovacie procedúry vytvorené a zostavené do testovacích sád, je možné tieto testovacie sady radiť do harmonogramu vykonávania definujúceho poradie spúšťania.

Pri stanovení priorít testovacích prípadov je možné zohľadniť rôzne faktory. Najčastejšie používané stratégie prioritizácie testovacích prípadov sú nasledujúce:

- **Prioritizácia na základe rizík**, kedy poradie vykonávania testov vychádza z výsledkov analýzy rizík (pozri *kapitolu 5.2.3*). Najprv sa vykonajú testovacie prípady pokrývajúce najdôležitejšie riziká.
- **Prioritizácia na základe pokrytia**, kedy je poradie vykonávania testov založené na určitom pokrytí (napr. pokrytie príkazov). Testovacie prípady dosahujúce najvyššie pokrytie sú vykonávané ako prvé. V inom variante (nazývanom prioritizácia dodatočného pokrytia) sa najprv vykoná testovací prípad s najvyšším pokrytím. Každý nasledujúci testovací prípad je ten, ktorý dosiahne najvyššie pokrytie dodatočnej fukcionality.
- **Prioritizácia na základe požiadaviek**, kedy poradie vykonávania testov vychádza z priorít požiadaviek trasovateľných späť k zodpovedajúcim testovacím prípadom. Priority požiadaviek definujú zainteresované strany, testovacie prípady súvisiace s najdôležitejšími požiadavkami sú vykonávané ako prvé.

V ideálnom prípade by mali byť testovacie prípady zoradené na vykonávanie na základe úrovne ich priority, napríklad pomocou jednej z vyššie uvedených stratégií prioritizácie. Tento postup však nemusí fungovať, ak medzi testovacími prípadmi alebo testovanými funkcionalitami existujú závislosti. Ak testovací prípad s vyššou prioritou závisí od testovacieho prípadu s nižšou prioritou, musí byť najskôr vykonaný testovací prípad s nižšou prioritou.

Poradie vykonávania testov musí tiež zohľadňovať dostupnosť zdrojov, napr. požadovaných testovacích nástrojov, testovacieho prostredia alebo osôb, ktoré môžu byť k dispozícii iba po určitú dobu.

### 5.1.6 Testovacia pyramída

Testovacia pyramída je model, ktorý ukazuje, že rôzne testy môžu mať rôznu granularitu. Predstavuje pomôcku pre tím pri automatizácii testov a smerovaní testovacieho úsilia poukázaním na to, ako sú rozličné ciele dosahované prostredníctvom rôznej miery automatizácie testovania.

Vrstvy pyramídy predstavujú skupiny testov. Čím vyššia vrstva, tým nižšia granularita testu, menšia izolácia testu a dlhšia doba vykonávania testu. Testy v spodnej vrstve sú malé, izolované, rýchle a overujú malú časť funkcionality, takže na dosiahnutie rozumného pokrytia je ich obvykle potrebné veľké množstvo. Horná vrstva predstavuje komplexné vysokoúrovňové E2E (end-to-end) testy. Tie sú všeobecne pomalšie ako testy z nižších vrstiev a obvykle overujú veľkú časť funkcionality, takže na dosiahnutie rozumného pokrytia je ich obvykle potrebných len niekoľko. Počet a pomenovanie jednotlivých vrstiev sa môže líšiť, napríklad pôvodný model testovacej pyramídy (Cohn, 2010) definuje tri vrstvy (testy komponentov, testy služieb a testy používateľského rozhrania). Iné modely definujú jednotkové testy (testy komponentov), integračné testy komponentov (integračné testy komponentov) a end-to-end testy. Všeobecne možno použiť aj iné úrovne testovania (pozri *kapitolu 2.2.1*).

### 5.1.7 Kvadranty testovania

Kvadranty testovania definované Brianom Marickom (Marick, 2003), (Crispin et al., 2009), dávajú do súvislosti úrovne testovania s príslušnými typmi testovania, činnosťami, technikami testovania a pracovnými produktmi v agilnom vývoji softvéru. Model je pomôckou pre manažment testovania pri vizualizácii týchto vzťahov s cieľom zaisťovať, že všetky vhodné typy a úrovne testovania sú zahrnuté do SDLC, a pre pochopenie toho, že niektoré typy testovania sú pre určité úrovne testovania relevantnejšie ako iné. Poskytuje spôsob, ako od seba rozlíšiť a popísať typy testovania všetkým zainteresovaným stranám vrátane vývojárov, testerov a zástupcov biznisu.

V tomto modeli os Y rozlišuje testy zamerané na biznis alebo na technológiu a os X potom testy podporujúce tím (t. j. tie, ktoré pomáhajú usmerňovať vývojové aktivity) alebo revidujúce produkt (t. j. také, ktoré pomáhajú merať jeho správanie oproti očakávaniam). Kombinácia týchto dvoch pohľadov (osí) určuje štyri kvadranty:

- **Kvadrant Q1 (testy zamerané na technológiu, podporujúce tím)**. Tento kvadrant obsahuje testy komponentov a integračné testy komponentov. Tieto testy by mali byť automatizované a zahrnuté do procesu CI.
- **Kvadrant Q2 (testy zamerané na biznis, podporujúci tím)**. Tento kvadrant obsahuje funkcionálne testy, príklady, testy používateľských scenárov, UX prototypy, testovanie rozhrania (API) a simulácie. Tieto testy overujú akceptačné kritériá a môžu byť manuálne aj automatizované.
- **Kvadrant Q3 (testy zamerané na biznis, revidujúce produkt)**. Tento kvadrant obsahuje prieskumné testovanie, testovanie použiteľnosti a používateľské akceptačné testovanie. Tieto testy sú používateľsky orientované a často manuálne.
- **Kvadrant Q4 (testy zamerané na technológiu, revidujúce produkt)**. Tento kvadrant obsahuje smoke testy a nefunkcionálne testy (okrem testov použiteľnosti). Tieto testy sú často automatizované.

## 5.2 Manažment rizík

Organizácie čelia mnohým interným a externým faktorom prinášajúcim neistotu v tom, či a kedy dosiahnu svoje ciele (*Management rizik – Principy a směrnice*, 2018). Manažment rizík im umožňuje zvýšiť pravdepodobnosť dosiahnutia týchto cieľov, zlepšiť kvalitu ich produktov a zvýšiť dôveru zainteresovaných strán.

Hlavnými činnosťami v oblasti manažmentu rizík sú:

- **analýza rizík** (obsahuje identifikáciu a ohodnotenie rizík, pozri *kapitolu 5.2.3*),
- **riadenie rizík** (obsahuje zmierňovanie a monitoring rizík, pozri *kapitolu 5.2.4*).

Prístup k testovaniu, pri ktorom sú testovacie činnosti vyberané, prioritizované a riadené na základe analýzy rizík spoločne s riadením rizík, sa nazýva testovanie založené na rizikách.

### 5.2.1 Definícia rizika a jeho atribúty

Riziko je možná udalosť, nebezpečenstvo, hrozba alebo situácia, ktorej výskyt má nepriaznivý vplyv. Riziko je možné charakterizovať dvoma faktormi:

- pravdepodobnosť rizika – pravdepodobnosť výskytu rizika vyjadrená v percentách alebo v intervale (0,1), t. j. hodnoty 0 a 1 nie sú súčasťou intervalu,
- vplyv rizika (škoda) – dôsledky výskytu takejto udalosti.

Tieto dva faktory vyjadrujú úroveň rizika, ktorá je jeho mierou (metrikou). Čím vyššia je úroveň rizika, tým dôležitejšie je jeho ošetrenie.

### 5.2.2 Projektové a produktové riziká

Pri testovaní softvéru sa všeobecne zaoberáme dvoma typmi rizík – projektovými rizikami a produktovými rizikami.

**Projektové riziká** sa týkajú manažmentu a riadenia projektu. Medzi projektové riziká patria:

- organizačné problémy (napr. oneskorenie pri dodávaní pracovných produktov, nepresné odhady, znižovanie nákladov),
- problémy s ľuďmi (napr. nedostatočné zručnosti, konflikty, komunikačné problémy, nedostatok zamestnancov),
- technické problémy (napr. narastajúci rozsah projektu, zlá podpora z pohľadu nástrojov),
- problémy s dodávateľmi (napr. zlyhanie dodávky od tretej strany, krach dodávateľa).

Projektové riziká, ak sa vyskytnú, môžu mať vplyv na harmonogram, rozpočet alebo rozsah projektu, čo ovplyvňuje schopnosť projektu dosiahnuť svoje ciele.

**Produktové riziká** súvisia s kvalitatívnymi charakteristikami produktu (napr. popísanými v modeli kvality (*Systémové a softwarové inženýrství – Požadavky a hodnocení kvality systémů a softwaru (SQuaRE), Modely kvality systémů a softwaru*, 2011)). Medzi príklady produktových rizík patria:

- chýbajúca alebo nesprávna funkcionalita,
- nesprávne výpočty,

- chyby pri behu (runtime errors),
- zlá architektúra,
- neefektívne algoritmy,
- nedostatočná doba odozvy,
- zlý používateľský zážitok,
- bezpečnostné zraniteľnosti.

Produktové riziká, ak sa vyskytnú, môžu mať rôzne negatívne dôsledky, napríklad:

- nespokojnosť používateľov,
- strata príjmov, dôvery, povesti,
- škody spôsobené tretím stranám,
- vysoké náklady na údržbu,
- preťaženie helpdesku,
- trestnoprávne postihy,
- fyzické poškodenie, zranenie, v extrémnych prípadoch smrť.

### 5.2.3 Analýza produktových rizík

Z hľadiska testovania je cieľom analýzy produktových rizík poskytnúť povedomie o produktových rizikách so zámerom nasmerovať testovacie úsilie tak, aby došlo k minimalizácii miery zostávajúcich rizík. V ideálnom prípade začína analýza produktových rizík v ranej fáze SDLC. Analýza produktových rizík sa skladá z identifikácie a ohodnotenia rizík.

**Identifikácia rizík** spočíva vo vytvorení komplexného zoznamu rizík. Zainteresované strany môžu identifikovať riziká pomocou rôznych techník a nástrojov, napr. brainstormingu, workshopov, rozhovorov alebo diagramov príčin a následkov.

**Ohodnotenie rizík** zahŕňa kategorizáciu identifikovaných rizík, určenie ich pravdepodobnosti, dopadu a úrovne, stanovenie priorít a navrhnutie spôsobov ich riešenia. Kategorizácia pomáha pri priraďovaní zmierňujúcich opatrení, pretože zvyčajne možno riziká patriace do rovnakej kategórie zmierniť podobnými činnosťami.

Ohodnotenie rizík môže využívať kvantitatívny alebo kvalitatívny prístup, prípadne ich kombináciu. Pri kvantitatívnom prístupe sa úroveň rizika vypočíta ako násobok pravdepodobnosti rizika a dopadu rizika. Pri kvalitatívnom prístupe možno úroveň rizika stanoviť pomocou matice rizík.

Analýza produktových rizík môže ovplyvniť dôkladnosť a rozsah testovania. Jej výsledky sa používajú na:

- určenie rozsahu testovania, ktoré sa má vykonať,
- určenie konkrétnych úrovní testovania a návrhu typov testovania, ktoré sa majú vykonať,
- určenie techník testovania, ktoré majú byť použité,
- určenie pokrytia, ktoré sa má dosiahnuť,
- odhadu úsilia potrebného na testovanie vyžadovaného pre každú úlohu,

- stanovenie priorít testovania s cieľom čo najrýchlejšieho nachádzania kritických defektov,
- určenie, či by sa mohli na zníženie rizika použiť aj iné činnosti ako testovanie.

### 5.2.4 Riadenie produktových rizík

Riadenie produktových rizík zahŕňa všetky opatrenia, ktoré sú prijaté v reakcii na identifikované a ohodnotené produktové riziká. Skladá sa zo zmierňovania rizík a monitorovania rizík.

Obsahom **zmierňovania rizík** je zavedenie opatrení navrhnutých pri ohodnotení rizík s cieľom znížiť úroveň rizika. Akonáhle je riziko analyzované, je možné naň reagovať niekoľkými spôsobmi, napr. jeho zmiernením pomocou testovania, prijatím rizika, prenosom rizika alebo záložným plánom (Veenendaal, 2012). Opatrenia, ktoré možno prijať na zmiernenie produktových rizík prostredníctvom testovania, sú nasledujúce:

- výber testerov so skúsenosťami a zručnosťami vhodnými pre daný typ rizika,
- použitie vhodnej úrovne nezávislosti testovania,
- vykonanie revízií a statickej analýzy,
- použitie vhodných techník testovania a úrovní pokrytia,
- použitie vhodných typov testovania zameraných na dotknuté kvalitatívne charakteristiky,
- vykonanie dynamického testovania (vrátane regresného testovania).

Cieľom **monitorovania rizík** je zisťovanie či sú zmierňujúce opatrenia efektívne, získanie ďalších informácií pre zlepšenie ohodnotenia rizík a identifikácia novo vznikajúcich rizík.

## 5.3 Monitorovanie, riadenie a dokončenie testovania

**Monitorovanie testovania** sa zaoberá zhromažďovaním informácií o testovaní. Tieto informácie sa využívajú na posúdenie postupu prác pri testovaní a na stanovenie, či sú naplnené výstupné kritériá testovania alebo dokončené testovacie úlohy spojené s výstupnými kritériami (ako napr. splnenie cieľov na pokrytie produktových rizík, požiadaviek alebo akceptačných kritérií).

**Riadenie testovania** využíva informácie z monitorovania testovania na to, aby boli formou usmernenia poskytnuté pokyny a nevyhnutné nápravné opatrenia na dosiahnutie čo najúčinnejšieho a najefektívnejšieho testovania. Medzi príklady takýchto usmernení patria:

- prehodnotenie priorít testov pri výskyte identifikovaného rizika,
- prehodnotenie, či v dôsledku zmien (prepracovania) spĺňa daná položka testovania vstupné alebo výstupné kritériá,
- úprava harmonogramu testovania s ohľadom na oneskorenie v dodaní testovacieho prostredia,
- pridávanie nových zdrojov tam, kde sú potrebné a vtedy, keď sú potrebné.

**Pri dokončení testov** sa zhromažďujú dáta z dokončených testovacích činností za účelom konsolidácie skúseností, testvéru a ďalších dôležitých informácií. K činnostiam súvisiacim s dokončením testovania dochádza v rámci projektových míľnikov ako sú dokončenie úrovne testovania, dokončenie iterácie pri agilnom vývoji, dokončenie (alebo zrušenie) testovacieho projektu, vydanie softvérového systému alebo dokončenie servisného vydania (maintenance release).

### 5.3.1 Metriky používané v testovaní

Testovacie metriky sú zhromažďované s cieľom ukazovať postup voči plánovanému harmonogramu a rozpočtu, aktuálnu kvalitu testovaného objektu a efektivitu testovacích činností s ohľadom na ciele projektu alebo iterácie. Monitorovanie testovania zhromažďuje rôzne metriky na podporu riadenia a dokončenia testovania.

Medzi typické testovacie metriky patria:

- metriky o postupe projektu (napr. dokončenie úlohy, využitie zdrojov, úsilie potrebné na testovanie),
- metriky o postupe testovania (napr. postup implementácie testovacích prípadov, postup prípravy testovacích prostredí, počet spustených / nespustených testovacích prípadov, počet vykonaných testov, ktoré prešli / zlyhali, doba vykonania testu),
- metriky týkajúce sa kvality produktu (napr. dostupnosť, doba odozvy, stredná doba medzi poruchami),
- metriky nad defektmi (napr. počet a priority zistených / opravených defektov, hustota defektov - defect density, percento odhalených defektov – defect detection percentage, DPP),
- metriky nad rizikami (napr. miera zostávajúcich rizík),
- metriky pokrytia (napr. pokrytie požiadaviek, pokrytie kódu),
- metriky týkajúce sa nákladov (napr. náklady na testovanie, náklady na kvalitu v rámci organizácie).

### 5.3.2 Účel, obsah a cieľové skupiny správ z testovania

Cieľom podávania správ o testovaní je zhrnutie a komunikácia informácií z testovania počas jeho priebehu a po ňom. Správy o postupe testovania podporujú priebežné riadenie testovania. Musia poskytovať dostatok informácií na vykonanie zmien v harmonograme testovania, zdrojoch alebo plánu testovania, ak sú tieto zmeny potrebné z dôvodu odchýlky od plánu alebo zmeny okolností. Správy o dokončení testovania sumarizujú určitú fázu testovania (napr. úroveň testovania, testovací cyklus, iteráciu) a môžu poskytnúť informácie pre následné testovanie.

Počas monitorovania a riadenia testovania vytvára testovací tím správy o postupe testovania s cieľom poskytovať zainteresovaným stranám informácie. Správy o postupe testovania sú obvykle generované pravidelne (napr. denne, týždenne atď.) a obsahujú:

- testovacie obdobie,
- postup prác pri testovaní (napr. náskok alebo oneskorenie oproti harmonogramu) vrátane všetkých významných odchýlok,
- prekážky pri testovaní a ich riešenia,
- testovacie metriky (pozri *kapitolu 5.3.1*),
- nové a zmenené riziká počas testovacieho obdobia,
- testovanie plánované pre nasledujúce obdobie.

Správa o dokončení testovania sa pripravuje v priebehu fázy dokončenia testovania, kedy je projekt, úroveň testovania alebo typ testovania ukončený, a kedy sú (v ideálnom prípade) splnené stanovené

výstupné kritériá. Táto správa využíva informácie z čiastkových správ o postupe testovania a dopĺňa ich o ďalšie dáta. Typické správy o dokončení testovania obsahujú:

- zhrnutie testov,
- vyhodnotenie testovania a kvality produktu na základe pôvodného plánu testovania (t. j. cieľov testovania a výstupných kritérií),
- odchýlky od plánu testovania (napr. rozdiely oproti plánovanému harmonogramu, trvaniu a náročnosti),
- prekážky pri testovaní a ich riešenia,
- testovacie metriky založené na správach o postupe testovania,
- neošetrené riziká, nevyriešené defekty,
- získané skúsenosti (lessons learned), ktoré sú relevantné pre testovanie.

Rôzne cieľové skupiny potrebujú v týchto správach rôzne informácie a ovplyvňujú tým mieru formálnosti a frekvenciu podávania správ. Podávanie správ o postupe prác pri testovaní ostatným členom rovnakého tímu je väčšinou časté a neformálne, zatiaľ čo podávanie správ o dokončení testovania z ukončeného projektu sa vykonáva typicky iba raz, a to podľa definovanej šablóny.

V norme ISO/IEC/IEEE 29119-3 (*Softwarové a systémové inženýrství – Testování softwaru – Část 3: Dokumentace testování* , 2021) možno nájsť šablóny a príklady správ o postupe testovania (nazývané správy o stave testovania) a správ o dokončení testovania.

### 5.3.3 Komunikovanie stavu testovania

Optimálny spôsob komunikovania stavu testovania sa líši v závislosti na potrebách manažmentu testovania, stratégiách testovania v organizácii, regulatórnych normách alebo v prípade samoorganizovaných tímov (pozri *kapitolu 1.5.2*) na samotnom tíme. Medzi možnosti komunikácie patria:

- verbálna komunikácia s členmi tímu a ďalšími zainteresovanými stranami,
- dashboardy (napr. CI/CD dashboard, tabule s úlohami a grafy burn-down),
- elektronické komunikačné kanály (napr. e-mail, chat),
- online dokumentácia,
- formálne správy z testovania (pozri *kapitolu 5.3.2*).

Je možné použiť jednu z uvedených možností alebo ich kombináciu. Formálnejšia komunikácia môže byť vhodnejšia pre distribuované tímy, kde nie je priama osobná komunikácia vždy možná kvôli geografickej vzdialenosti alebo časovým posunom. Obvykle sa rôzne zainteresované strany zaujímajú o iný typ informácií, preto by mala byť komunikácia zodpovedajúcim spôsobom prispôsobená.

## 5.4 Konfiguračný manažment

V oblasti testovania je konfiguračný manažment disciplína, ktorá slúži na identifikáciu, riadenie a sledovanie pracovných produktov. Konfiguračnými položkami môžu byť akékoľvek plány testovania, stratégie testovania, testovacie podmienky, testovacie prípady, skripty, výsledky testov, protokoly testov (test logs) a správy z testovania.

V prípade komplexnej konfiguračnej položky (napr. testovacie prostredie) je možné v rámci konfiguračného manažmentu zaznamenávať čiastkové položky, z ktorých sa táto komplexná položka skladá, ich vzťahy a verzie. V momente, keď je konfiguračná položka schválená pre testovanie, stáva sa súčasťou tzv. baseline a je možné ju meniť iba prostredníctvom formálneho procesu na riadenie zmien.

V prípade vytvorenia novej baseline je úlohou konfiguračného manažmentu uchovávať záznamy o všetkých zmenených konfiguračných položkách. Pokiaľ je napríklad nutné reprodukovať predchádzajúce výsledky testov, je možné vrátiť sa k predchádzajúcej baseline.

Konfiguračný manažment ako podporná disciplína testovania zaisťuje nasledujúce:

- Všetky konfiguračné položky vrátane položiek testovania (jednotlivých častí testovaného objektu) sú jednoznačne identifikované, ich verzie sú riadené, všetky zmeny sú evidované a vzťahy s inými konfiguračnými položkami sú zaznamenané tak, aby bolo možné zaistiť sledovateľnosť v priebehu celého procesu testovania.
- V dokumentácii testovania vedú na všetky identifikované dokumenty a softvérové položky jednoznačné odkazy.

Kontinuálna integrácia, kontinuálne dodávanie, kontinuálne nasadzovanie a s týmito procesmi súvisiace testovanie sú obvykle implementované ako súčasť automatizovanej DevOps pipeline (pozri *kapitolu 2.1.4*), ktorej súčasťou je obvykle aj automatizovaný konfiguračný manažment.

## 5.5 Manažment defektov

Keďže jedným z hlavných cieľov testovania je nájdenie defektov, zavedenie procesu manažmentu defekt je nevyhnutné. Napriek tomu, že sa hovorí o "defektoch", platí, že nahlásené anomálie môžu byť skutočné defekty alebo aj niečo iné (napr. chybné výsledky testov alebo zmenové požiadavky). Toto je vyhodnocované v rámci spracovania reportu o defekte. Anomálie môžu byť hlásené v ktorejkoľvek fáze SDLC a forma ich hlásenia závisí od použitého SDLC.

Ako základné minimum musí proces manažmentu defektov obsahovať pracovný postup (workflow) pre spracovanie jednotlivých anomálií od ich odhalenia až po ich uzavretie a pravidlá pre ich klasifikáciu. Tento pracovný postup obvykle pozostáva z činností, ktorých cieľom je zaznamenávať nahlásené anomálie, analyzovať ich a klasifikovať, rozhodnúť o vhodnej reakcii (typicky opraviť alebo neriešiť) a uzavretie reportu o defekte. Tento proces musia dodržiavať všetky zúčastnené strany.

Podobným spôsobom je vhodné riešiť aj defekty zistené statickým testovaním (najmä statickou analýzou).

Platí, že nie všetky nahlásené anomálie musia byť nutne skutočné defekty (napr. falošne pozitívne výsledky testov alebo zmenové požiadavky). Toto vyhodnotenie je vykonávané v rámci spracovania reportu o defekte. Reporty o defekte majú obvykle nasledujúce ciele:

- Poskytovať osobám zodpovedným za spracovanie a riešenie hlásených defektov dostatočné informácie pre vyriešenie problému.

- Poskytovať spôsob sledovania kvality pracovného produktu.
- Navrhovať nápady na zlepšovanie procesov vývoja a testovania.

Report o defekte zaznamenaná počas dynamického testovania obvykle obsahuje:

- jedinečný identifikátor,
- názov obsahujúci krátke zhrnutie reportovanej anomálie,
- dátum zistenia anomálie, reportujúca organizácia a autor (vrátane roly autora),
- identifikáciu testovaného objektu a testovacieho prostredia,
- kontext defektu (napr. vykonaný testovací prípad, vykonaná testovacia činnosť, fáza SDLC a ďalšie relevantné informácie ako napr. použitá technika testovania, použitý kontrolný zoznam alebo použité testovacie dáta),
- popis zlyhania umožňujúci reprodukciu a vyriešenie vrátane krokov, ktoré anomáliu vyvolali, všetkých relevantných protokolov testov (test logs), záloh databáz, snímok obrazovky alebo videozáznamov,
- očakávané a skutočné výsledky,
- závažnosť defektu (mieru dopadu) na záujmy zainteresovaných strán alebo na požiadavky,
- naliehavosť / prioritu opravy,
- stav defektu (napr. otvorený, odložený, duplicitný, čakajúci na opravu, čakajúci na konfirmačné testovanie, znovu otvorený, uzavretý, zamietnutý),
- odkazy (napr. na testovací prípad).

Niektoré z týchto informácií môžu byť automaticky vkladané nástrojom pre manažment defektov (napr. identifikátor, dátum, autor a počiatočný stav). Šablóny dokumentov pre report o defekte a príklady reportov o defekte možno nájsť v norme STN ISO/IEC/IEEE 29119-3 (*Softwarové a systémové inženýrství – Testování softwaru – Část 3: Dokumentace testování* , 2021). Norma však používa termín správa o incidente.

# 6 Testovacie nástroje – 20 minút

**Kľúčové slová**

automatizácia testov

**Študijné ciele:**

## 6.1 Nástroje na podporu testovania

FL-6.1.1 (K2) Vysvetliť, ako rôzne typy testovacích nástrojov podporujú testovanie.

## 6.2 Prínosy a riziká automatizácie testov

FL-6.2.1 (K1) Zapamätať si prínosy a riziká automatizácie testov.

## 6.1 Nástroje na podporu testovania

Testovacie nástroje podporujú a uľahčujú mnoho testovacích aktivít. Príklady okrem iného zahŕňajú:

- nástroje pre manažment testovania – zvyšujú efektivitu testovacieho procesu uľahčením správy SDLC, požiadaviek, testov, defektov, konfigurácií,
- nástroje na statické testovanie – podporujú testera pri vykonávaní revízií a statickej analýzy,
- nástroje na návrh a implementáciu testov – uľahčujú vytváranie testovacích prípadov, testovacích dát a testovacích procedúr,
- nástroje na vykonávanie testov a meranie ich pokrytia – uľahčujú automatizované vykonávanie testov a meranie pokrytia,
- nástroje na nefunkcionálne testovanie – umožňujú testerovi vykonávať nefunkcionálne testovanie, ktoré je ťažké alebo nemožné vykonať manuálne,
- nástroje DevOps – podporujú pipeline v DevOps, sledovanie pracovných tokov, procesy automatizovaného zostavovania (automated build), CI/CD,
- nástroje na spoluprácu – uľahčujú komunikáciu,
- nástroje podporujúce škálovateľnosť a štandardizáciu nasadenia (napr. virtuálne počítače, nástroje na prácu s kontajnermi),
- akýkoľvek iný nástroj, ktorý pomáha pri testovaní (napr. tabuľkový procesor je v kontexte testovania testovacím nástrojom).

## 6.2 Prínosy a riziká automatizácie testov

Samotné obstaranie nástroja nezaručuje úspech. Každý nový nástroj si bude vyžadovať úsilie na dosiahnutie skutočných a trvalých prínosov (napr. na zavedenie nástroja, údržbu a školenie). Existujú aj určité riziká, ktoré je potrebné analyzovať a zmierniť.

Medzi potenciálne prínosy použitia automatizácie testov patria:

- Úspora času znížením opakovanej manuálnej práce (napr. vykonávanie regresných testov, opätovné zadávanie rovnakých testovacích dát, porovnávanie očakávaných výsledkov so skutočnými výsledkami a kontrola kódu oproti štandardom kódovania).
- Predchádzanie jednoduchým ľudským chybám vďaka väčšej konzistentnosti a opakovateľnosti (napr. testy sú dôsledne odvodené z požiadaviek, testovacie dáta sú vytvorené systematickým spôsobom a testy sú vykonávané nástrojom v rovnakom poradí s rovnakou frekvenciou).
- Objektívnejšie hodnotenie (napr. pokrytie) a vykonávanie opatrení, ktoré sú pre človeka príliš komplikované na odvodenie.
- Jednoduchší prístup k informáciám o testovaní pre podporu manažmentu testovania a podávanie správ o testovaní (napr. štatistiky, grafy a súhrnné dáta o priebehu testov, počte defektov a dĺžke vykonávania testov).
- Skrátenie času vykonávania testov na zabezpečenie skoršieho odhalenia defektov, rýchlejšej spätnej väzby a rýchlejšieho uvedenia na trh.

- Viac času pre testerov na navrhovanie nových, dôkladnejších a efektívnejších testov.

Potenciálne riziká používania automatizácie testov zahŕňajú:

- Nerealistické očakávania týkajúce sa prínosov nástroja (vrátane funkčnosti a jednoduchosti používania).
- Nepresné odhady času, nákladov a úsilia potrebného na zavedenie nástroja, udržiavanie testovacích skriptov a zmenu existujúceho procesu manuálneho testovania.
- Používanie testovacieho nástroja, keď je vhodnejšie manuálne testovanie.
- Prílišné spoliehanie sa na nástroj, napr. v zmysle ignorovania potreby ľudského kritického myslenia.
- Závislosť od dodávateľa nástroja, ktorý môže ukončiť svoju činnosť, prestať nástroj vyvíjať, predať nástroj inému dodávateľovi alebo poskytovať nedostatočnú podporu (napr. odpovede na otázky, aktualizácie a opravy chýb).
- Používanie open-source nástroja, ktorého vývoj môže byť zastavený, čo znamená nedostupnosť ďalších aktualizácií, alebo jeho vnútorné komponenty môžu vyžadovať pomerne časté aktualizácie v rámci ďalšieho vývoja.
- Automatizačný nástroj nie je kompatibilný s vývojovou platformou.
- Výber nevhodného nástroja, ktorý nespĺňa regulatórne požiadavky a/alebo bezpečnostné normy.

# 7 Príloha A – Študijné ciele a kognitívne úrovne znalostí

Na účely týchto učebných osnov sú definované študijné ciele. Každá téma v osnovách môže byť preskúšaná podľa zodpovedajúceho študijného cieľa. Študijné ciele začínajú vždy slovesom, ktoré zodpovedá jeho kognitívnej úrovni znalostí. Kognitívne úrovne vychádzajú z (Anderson et al., 2001).

**Úroveň 1: Zapamätať si (K1)** – kandidát je schopný zapamätať si, určiť alebo vybaviť si daný termín alebo koncept.

**Slovesá:** identifikovať, vybaviť si, zapamätať si, určiť.

**Príklady:**

- „Identifikovať typické ciele testovania.“
- „Zapamätať si koncept testovacej pyramídy.“
- „Určiť, akú pridanú hodnotu má prítomnosť testerov pre plánovanie iterácií a plánovanie vydania.“

**Úroveň 2: Porozumieť (K2)** – Kandidát dokáže označiť príčiny alebo vysvetlenia k výrokom vzťahujúce sa k téme, dokáže zhrnúť, porovnať, kategorizovať a uviesť príklady pre daný koncept testovania.

**Slovesá:** kategorizovať, porovnať, uviesť odlišnosti, rozlíšiť, odlíšiť, ilustrovať na príkladoch, vysvetliť, uviesť príklady, interpretovať, zhrnúť.

**Príklady:**

- „Kategorizovať rôzne možnosti na písanie akceptačných kritérií.“
- „Porovnať rôzne roly v testovaní“ (hľadajte podobnosti a/alebo rozdiely).
- „Odlíšiť projektové riziká od produktových rizík“ (umožňuje diferencovať pojmy).
- „Ilustrovať na príkladoch účel a obsah plánu testovania.“
- „Vysvetliť vplyv kontextu na proces testovania.“
- „Zhrnúť aktivity procesu revízií.“

**Úroveň 3: Použiť (K3)** – kandidát je schopný aplikovať daný postup, pokiaľ je konfrontovaný s obdobnou úlohou, alebo zvoliť správny postup a aplikovať ho na daný kontext.

**Slovesá:** aplikovať, implementovať, pripraviť, použiť.

**Príklady:**

- „Aplikovať prioritizáciu testovacieho prípadu“ (malo by odkazovať na postup, techniku, proces, algoritmus atď.).
- „Pripraviť report o defekte.“
- „Použiť techniku analýzy hraničných hodnôt na odvodenie testovacích prípadov.“

# 8 Príloha B – Poznámky k vydaniu

Učebné osnovy ISTQB® základnej úrovne (ISTQB® Foundation Level Syllabus) v4.0 sú významnou aktualizáciou, ktorá vychádza z predchádzajúcej verzie 3.1.1 osnov základnej úrovne a z učebných osnov Agile Tester 2014. Z tohto dôvodu neexistujú detailné poznámky k vydaniu (release notes) na úrovni jednotlivých kapitol a sekcií a prehľad zásadných zmien je uvedený v tejto kapitole.

Okrem toho poskytuje ISTQB® (v samostatných poznámkach k vydaniu) trasovateľnosť medzi študijnými cieľmi (LO) učebných osnov základnej úrovne vo verzii 3.1.1, študijnými cieľmi učebných osnov Agile Tester verzia 2014 a študijnými cieľmi učebných osnov základnej úrovne vo verzii 4.0, okrem toho sa zdôraznilo, ktoré študijné ciele boli pridané, aktualizované alebo odstránené.

V čase, keď boli tieto učebné osnovy vytvorené (2022 – 2023) bolo evidovaných viac ako 1 000 000 osôb vo viac ako 100 krajinách, ktoré absolvovali skúšku základnej úrovne. Celosvetovo existuje viac ako 800 000 certifikovaných testerov. Za predpokladu, že si všetci z nich prečítali učebné osnovy základnej úrovne s cieľom zložiť skúšku, je možné povedať, že ide pravdepodobne o najčítanejší dokument v oblasti testovania vôbec. Táto aktualizácia je vytvorená tak, aby tieto hodnoty rešpektovala a tiež, aby prispela k zlepšeniu názorov na úroveň kvality, ktorú ISTQB® prináša globálnej testovacej komunite.

V tejto verzii boli všetky študijné ciele upravené tak, aby boli atomické, a aby bola vytvorená trasovateľnosť 1:1 medzi študijnými cieľmi a kapitolami učebných osnov. Inými slovami, učebné osnovy neobsahujú kapitoly a obsah, ku ktorým neexistuje študijný cieľ. Cieľom je uľahčiť čítanie, porozumenie, učenie a preklad s dôrazom na zvýšenie praktickej užitočnosti a rovnováhy medzi znalosťami a zručnosťami.

V tejto verzii došlo k nasledujúcim zmenám:

- Zmenšenie celkovej veľkosti osnov. Učebné osnovy nie sú učebnice, ale dokument, ktorý slúži na načrtnutie základnej kostry akéhokoľvek úvodného kurzu o testovaní softvéru vrátane informácií o tom, aké témy by mali byť pokryté a na akej úrovni. To znamená nasledujúce:
  - Vo väčšine prípadov sú z textu vylúčené príklady. Úlohou poskytovateľa školenia je potom tieto príklady (vrátane praktických cvičení) poskytnúť počas školenia.
  - Bol dodržaný princíp „Veľkosť textu vzhľadom ku K-úrovni“, ktorý definuje maximálnu veľkosť textu pre študijné ciele na každej úrovni K (K1 = max. 10 riadkov, K2 = max. 15 riadkov, K3 = max. 25 riadkov).
- Zníženie počtu študijných cieľov v porovnaní s osnovami základnej úrovne (CTFL) verzie 3.1.1 a Agile Tester (CTFL-AT) verzie 2014:
  - 14 K1 cieľov (FL v3.1.1 mala 15 cieľov a AT 2014 6 cieľov),
  - 42 K2 cieľov (FL v3.1.1 mala 40 cieľov a AT 2014 13 cieľov),
  - 8 K3 cieľov (FL v3.1.1 mala 7 cieľov a AT 2014 8 cieľov).
- K dispozícii sú rozsiahlejšie odkazy na klasické a/alebo uznávané knihy a články o testovaní softvéru a súvisiacich témach.
- Hlavné zmeny v kapitole 1 (Základy testovania)
  - Bola rozšírená a vylepšená podkapitola o testovacích zručnostiach.
  - Bola pridaná podkapitola o tímovom prístupe (K1).

  - Podkapitola o nezávislosti testovania bola presunutá z kapitoly 5 do kapitoly 1.
- Hlavné zmeny v kapitole 2 (Testovanie v rámci SDLC)
  - Podkapitoly 2.1.1 a 2.1.2 boli prepracované a vylepšené, zároveň boli upravené zodpovedajúce študijné ciele.
  - Je kladený vyšší dôraz na techniky vývoja iniciovaného testami (K1), prístupu shift-left (K2) a retrospektívy (K2).
  - Pridaná nová podkapitola o testovaní v kontexte DevOps (K2).
  - Úroveň integračného testovania bola rozdelená do dvoch samostatných testovacích úrovní (testovanie integrácie komponentov a testovanie integrácie systému).
- Hlavné zmeny v kapitole 3 (Statické testovanie)
  - Podkapitola o technikách revízie spolu so zodpovedajúcim študijným cieľom „(K3) Použiť techniku revízie“ bola odstránená.
- Hlavné zmeny v kapitole 4 (Testovacia analýza a návrh testov)
  - Testovanie prípadov použitia bolo odstránené (ale je stále predmetom osnov Advanced Test Analyst).
  - Vyšší dôraz je kladený na techniky testovania založených na spolupráci (pridaný nový K3 študijný cieľ o použití ATDD na odvodenie testovacích prípadov a dva nové K2 ciele o užívateľských scenároch a akceptačných kritériách).
  - Testovanie a pokrytie rozhodnutí nahradené testovaním a pokrytím vetiev (po prvé, termín pokrytia vetiev sa v praxi používa viac; za druhé, rôzne štandardy definujú termín rozhodnutia odlišne od vetvy; po tretie, rieši to drobný, ale závažný nedostatok zo starých osnov základnej úrovne verzie 2018, kde je napísané, že „100% pokrytie rozhodnutia znamená 100% pokrytie príkazov“ – táto veta nie je pravdivá v prípade kódu bez podmienok alebo cyklov, teda bez bodov rozhodnutia).
  - Bola vylepšená podkapitola o hodnote testovania bielej skrinky.
- Hlavné zmeny v kapitole 5 (Manažment testovania)
  - Oddiel o testovacích stratégiách/prístupoch bol odstránený.
  - Pridaný nový K3 študijný cieľ o technikách odhadovania úsilia potrebného na testovanie.
  - Je kladený vyšší dôraz na všeobecne známe agilné koncepty a nástroje pri manažmente testovania: plánovanie iterácií a vydaní (K1), testovacia pyramída (K1) a kvadranty testovania (K2).
  - Oddiel o manažmente rizík je lepšie štruktúrovaný popisom štyroch hlavných činností: identifikácia rizík, posúdenie rizík, zmiernenie rizík a monitoring rizík.
- Hlavné zmeny v kapitole 6 (Testovacie nástroje)
  - Rozsah textu o automatizácii testov bol znížený z dôvodu príliš vysokej odbornosti nevhodnej pre učebné osnovy základnej úrovne.

– Podkapitola o výbere nástrojov, realizácii pilotných projektov a zavádzaní nástrojov do organizácie bola odstránená.

# 9 Referencie

## Štandardy

*Management rizik – Principy a směrnice*, 2018. Geneva, CH, 2018-12. Standard. International Organization for Standardization.

*Softwarové inženýrství – Procesy životního cyklu softwaru – Údržba*, 2010. Geneva, CH, 2010-03. Standard. International Organization for Standardization.

*Softwarové a systémové inženýrství – revize pracovních produktů*, 2017. Geneva, CH, 2017-03. Standard. International Organization for Standardization.

*Systémové a softwarové inženýrství – Požadavky a hodnocení kvality systémů a softwaru (SQuaRE), Modely kvality systémů a softwaru*, 2011. Geneva, CH, 2011-03. Standard. International Organization for Standardization.

*Softwarové a systémové inženýrství – Testování softwaru – Část 1: Obecné pojmy*, 2022. Geneva, CH, 2022-11. Standard. International Organization for Standardization.

*Softwarové a systémové inženýrství - Testování softwaru - Část 2: Testovací procesy*, 2021. Geneva, CH, 2021-04. Standard. International Organization for Standardization.

*Softwarové a systémové inženýrství – Testování softwaru – Část 3: Dokumentace testování*, 2021. Geneva, CH, 2021-04. Standard. International Organization for Standardization.

*Softwarové a systémové inženýrství – Testování softwaru – Část 4: Techniky testování*, 2021. Geneva, CH, 2021-11. Standard. International Organization for Standardization.

## Knihy

ADZIC, G., 2009. *Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing*. Neuri Limited. ISBN 9780955683619.

AMMANN, P.; OFFUTT, J., 2016. *Introduction to Software Testing 2nd Edition*. Cambridge University Press. ISBN 9781107172012.

ANDERSON, L.W.; KRATHWOHL, D.R., 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Longman. ISBN 9780801319037.

ANDREWS, M.; WHITTAKER, J.A., 2006. *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services*. Pearson Education. ISBN 9780321657510.

BECK, K., 2003. *Test-driven Development: By Example*. Addison-Wesley. Addison-Wesley signature series. ISBN 9780321146533.

BEIZER, B., 1990. *Software Testing Techniques*. Van Nostrand Reinhold. ISBN 9780442206727.

BOEHM, B.W., 1981. *Software Engineering Economics*. Prentice-Hall. Prentice-Hall advances in computing science and technology series. ISBN 9780138221225.

BUXTON, J.N.; RANDELL, B.; COMMITTEE, NATO Science, 1970. *Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969*. NATO Science Committee.

CHELIMSKY, D., 2010. *The RSpec Book: Behaviour-driven Development with RSpec, Cucumber, and Friends*. Pragmatic Bookshelf. Pragmatic Bookshelf Series. ISBN 9781934356371.

COHN, M., 2010. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley. A Mike Cohen signature book. ISBN 9780321579362.

COPELAND, L., 2004. *A Practitioner's Guide to Software Test Design*. Artech House. Artech House computing library A practitioner's guide to software test design. ISBN 9781580537322.

CRAIG, R.D.; JASKIEL, S.P., 2002. *Systematic Software Testing*. Artech House. Artech House ITS library. ISBN 9781580537926.

CRISPIN, L.; GREGORY, J., 2009. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley. A Mike Cohn signature book. ISBN 9780321534460.

FORGACS, I.; KOVACS, A., 2019. *Practical Test Design: Selection of traditional and automated test design techniques*. BCS, The Chartered Institute for IT. ISBN 9781780174723.

GÄRTNER, Markus, 2012. *ATDD by example: a practical guide to acceptance test-driven development*. Addison-Wesley.

GAWANDE, A., 2011. *The Checklist Manifesto: How to Get Things Right*. Metropolitan Books. ISBN 9780312430009.

HENDRICKSON, E., 2013. *Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing*. Pragmatic Bookshelf. The Pragmatic Bookshelf. ISBN 9781937785024.

HETZEL, W.C., 1988. *The Complete Guide to Software Testing*. QED Information Sciences. ISBN 9780894352423.

JEFFRIES, R.; ANDERSON, A.; HENDRICKSON, C., 2001. *Extreme Programming Installed*. Addison-Wesley. XP series. ISBN 9780201708424.

JORGENSEN, P., 2014. *Software Testing: A Craftsman's Approach, Fourth Edition*. Auerbach Publications.

KAN, S.H., 2003. *Metrics and Models in Software Quality Engineering*. Addison-Wesley. ISBN 9780201729153.

KANER, C.; FALK, J.; NGUYEN, H.Q., 1999. *Testing Computer Software*. Wiley. Bibliyografya ve İndeks. ISBN 9780471358466.

KIM, G.; HUMBLE, J.; DEBOIS, P.; WILLIS, J., 2016. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press. ITpro collection. ISBN 9781942788072.

MYERS, G.J.; SANDLER, C.; BADGETT, T., 2011. *The Art of Software Testing*. Wiley. ITPro collection. ISBN 9781118133156.

O'REGAN, G., 2019. *Concise Guide to Software Testing*. ISBN 978-3-030-28493-0. Dostupné z DOI: 10.1007/978-3-030-28494-7.

ROMAN, A., 2018. *Thinking-Driven Testing. The Most Reasonable Approach to Quality Control*. Springer Nature Switzerland.

VEENENDAAL, E. van, 2012. *The PRISMA Approach: Practical Risk-Based Testing*. UTN Publishers.

VROON, Michiel; BROEKMAN, Bart; KOOMEN, Tim; AALST, Leo van der, 2006. *TMap next, for result-driven testing*. UTN Publishers.

WATSON, A.H.; WALLACE, D.R.; MCCABE, T.J.; ASSOCIATES, McCabe &; STANDARDS, National Institute of; (U.S.), Technology, 1996. *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. U.S. Department of Commerce, Technology Administration, National Institute of Standards a Technology. NIST special publication, č. sv. 13. ISBN 9780160533815.

WESTFALL, L., 2016. *The Certified Software Quality Engineer Handbook*. Quality Press. ISBN 9781951058777.

WHITTAKER, J.A., 2003. *How to Break Software: A Practical Guide to Testing*. Addison Wesley. International computer science series. ISBN 9780201796193.

WHITTAKER, J.A., 2009. *Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design*. Pearson Education. ISBN 9780321647856.

## Články

BRYKCZYNSKI, Bill, 1999. A survey of software inspection checklists. *ACM SIGSOFT Software Engineering Notes*. Roč. 24, č. 1, s. 82.

ENDRES, Albert, 1975. An analysis of errors and their causes in system programs, s. 327–336.

MANNA, Zohar; WALDINGER, Richard, 1978. The logic of computer programming. *IEEE transactions on Software Engineering*. Č. 3, s. 199–229.

NIELSEN, Jakob, 1994. Enhancing the explanatory power of usability heuristics, s. 152–158.

SALMAN, Iflaah, 2016. Cognitive biases in software quality and testing, s. 823–826.

# 10 Ďalšie zdroje

GILB, T.; GRAHAM, D.; FINZI, S., 1993. *Software Inspection*. Addison-Wesley. Software engineering. ISBN 9780201631814.

KANER, C.; BACH, J.; PETTICHORD, B., 2011. *Lessons Learned in Software Testing: A Context-Driven Approach*. Wiley. ISBN 9781118080559.

MARICK, B., 2003. *Exploration through Example*. Dostupné tiež z: http://www.exampler.com/old-blog/2003/08/21.1.html#agile-testing-project-1.

PRESSMAN, R.S.; MAXIM, B.R., 2019. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education. ISBN 9781259872976.

WAKE, B., 2003. *INVEST in Good Stories, and SMART Tasks*. Dostupné tiež z: https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/.

WHITTAKER, J.A.; THOMPSON, H.H., 2004. *How to Break Software Security: Effective Techniques for Security Testing*. Pearson/Addison Wesley. ISBN 9780321194336.

WIEGERS, K.E., 2002. *Peer Reviews in Software: A Practical Guide*. Addison-Wesley. Addison-Wesley information technology series. ISBN 9780201734850.