Učební osnovy – Certifikovaný tester základní úrovně

verze 4.0

Czech and Slovak Quality Board

Upozornění o ochraně autorských práv

Historie změn (česká verze)

Verze	Datum	Poznámka
ISTQB® CTFL CZ 4.0.3	5.4.2024	Oprava překlepů. Oprava významu zkratek DDD a DDP. Změna názvu spolku na CaSQB.
ISTQB® CTFL CZ 4.0.2	30.12.2023	Oprava překlepů.
ISTQB® CTFL CZ 4.0.1	1.11.2023	Oprava překlepů. Sjednocení lexikální a stylistické úrovně.
ISTQB® CTFL CZ 4.0.0	6.9.2023	Oficiální veřejná verze.
ISTQB® CTFL CZ 4.0.0 Beta	10.8.2023	Nová verze dle CTFL v4.0. Neveřejná beta verze.
ISTQB® CTFL 2018 CZ 3.1.0	30.1.2020	Nová verze dle ISTQB® CTFL verze 3.1. Verzování přizpůsobeno verzování původních ISTQB® materiálů.
ISTQB® CTFL 2018 CZ 1.0.1	31.8.2019	Oprava překlepů a drobných formálních chyb.
ISTQB® CTFL 2018 CZ 1.0.0	4.2.2019	Oficiální vydání 1.0.0.
ISTQB® CTFL 2018 CZ Beta	9.1.2019	Beta verze pro členy a partnery CaSTB.

Historie změn (anglický originál)

Verze	Datum	Poznámka
CTFL v4.0	21.4.2023	CTFL v4.0 – General release version
CTFL v3.1.1	01.07.2021	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

Poděkování (ISTQB®)

Tento dokument byl formálně vydán Valným shromážděním ISTQB® dne 21.4.2023. Byl sepsán společným týmem pracovních skupin ISTQB® Foundation Level a Agile Working Groups ve složení: Laura Albert, Renzo Cerquozzi (místopředseda), Wim Decoutere, Klaudia Dussa-Zieger, Chintaka Indikadahena, Arnika Hryszko, Martin Klonk, Kenji Onishi, Michaël Pilaeten (předseda), Meile Posthuma, Gandhinee Rajkomar, Stuart Reid, Eric Riou du Cosquer (předseda), Jean-François Riverin, Adam Roman, Lucjan Stapp, Stephanie Ulrich (místopředsedkyně), Eshraka Zakaria. Tým děkuje Stuartu Reidovi, Patricii McQuaidové a Leanne Howardové za technickou revizi a reviznímu týmu a členským výborům za jejich návrhy a příspěvky.

Na revizi, přípomínkování a hlasování o těchto učebních osnovách se podíleli tito lidé: 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 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 Munzel, 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, Par Tatiana Tahidu Thahideva, 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 (verze 2018): Klaus Olsen (předseda), Tauhida Parveen (místopředseda), Rex Black (projektový manažer), Dani Almog, Debra Friedenberg, Rashed Karim, Johan Klintin, Vipul Kocher, Corne Kruger, Sunny Kwon, Judy McKay, Thomas Müller, Igal Levi, Ebbe Munk, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, Eshraka Zakaria, a Stevan Zivanovic. Tým děkuje týmu revidujících a všem členským výborům za návrhy k současným učebním osnovám.

ISTQB® Working Group Foundation Level (verze 2011): Thomas Müller (předseda), Debra Friedenberg. Hlavní tým děkuje týmu revidujících (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquer, Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal) a všem členským výborům za návrhy k současným učebním osnovám.

ISTQB® Working Group Foundation Level (verze 2010): Thomas Müller (předseda), Rahul Verma, Martin Klonk a Armin Beer. Hlavní tým děkuje týmu revidujících (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Tuula Pääkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veendendaal) a všem členským výborům za jejich připomínky.

ISTQB® Working Group Foundation Level (verze 2007): Thomas Müller (předseda), Dorothy Graham, Debra Friedenberg a Erik van Veendendaal. Hlavní tým děkuje týmu revidujících (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson a Wonil Kwon) a všem členským výborům za návrhy.

ISTQB® Working Group Foundation Level (verze 2005): Thomas Müller (předseda), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson a Erik van Veendendal. Hlavní tým děkuje týmu revidujících a všem členským výborům za návrhy.

Poděkování (CaSTB)

Překlad do českého jazyka, verze v4.0: David Janota, Petr Neugebauer a Lukáš Piška (překlad), Miroslav Renda a Pavel Herout (revize).

Překlad do českého jazyka, verze 2018 v3.1: David Janota, Petr Neugebauer a Miroslav Renda.

Překlad do českého jazyka, verze 2018 v1.0: David Janota, Petr Neugebauer a Miroslav Renda (překlad) a Karol Frühauf, Pavel Herout, Tereza Kuco, Martin Malaník, Lukáš Piška a Jana Zientková (revize).

Překlad do českého jazyka, verze 2011: David Janota, Petr Neugebauer, Jana Podpěrová, Gabor Puhalla, Lukáš Piška, Miroslav Renda a Jana Zientková.

Překlad do českého jazyka, verze 2007: Alexandra Alvarová, Róbert Dankanin, Petr Neugebauer, Jana Podpěrová, Jana Zientková.

Byť bylo snahou překladatelů docílit co nejvěrnějšího překladu původního anglického vydání, prostor pro zdokonalování v tak komplexním textu jistě je a stále bude. Postřehy a podněty čtenářů proto rádi uvítáme e-mailem na adrese translation@castb.org.

0 Úvod

0.1 Účel učebních osnov

Tyto učební osnovy tvoří základ pro mezinárodní kvalifikaci testování softwaru základní úrovně. ISTQB® poskytuje tyto učební osnovy takto:

1. Členským výborům pro překlad do daného národního jazyka a za účelem akreditace poskytovatelů školení. Členské výbory mohou přizpůsobit osnovy konkrétním potřebám svého jazyka a přidat odkazy na lokální publikace.
2. Certifikačním autoritám za účelem vytvoření zkouškových otázek v národním jazyce uzpůsobených studijním cílům pro tyto osnovy.
3. Poskytovatelům školení pro přípravu výukových kurzů a stanovení vhodných výukových metod.
4. Zájemcům o certifikaci za účelem přípravy na certifikační zkoušku, a to buď jako součást výukového kurzu nebo i samostatně.
5. Mezinárodní komunitě softwarových a systémových inženýrů za účelem podpory profese testování softwaru a systémů, a rovněž jako zdroj pro knihy a články.

0.2 Certifikovaný tester základní úrovně pro testování softwaru

Tato úroveň je určena každému, kdo se podílí na testování softwaru, což mohou být lidé v rolích testerů, analytiků testování, specialistů v oblasti testování, konzultantů, manažerů testování, softwarových vývojářů nebo jiných členů týmů. Je rovněž vhodná pro každého, kdo chce získat základní znalosti v oblasti testování softwaru, například pro vlastníky produktů (product owners), projektové manažery, manažery kvality, manažery vývoje softwaru, byznysové analytiky, IT ředitele a konzultanty na úrovni managementu. Získání certifikátu základní úrovně opravňuje jeho držitele k certifikacím vyšších úrovní v oblasti testování softwaru.

0.3 Kariérní cesta pro testery

Program ISTQB® podporuje odborníky v oblasti testování na všech úrovních jejich kariérního rozvoje a nabízí rozvoj znalostí jak do šířky, tak do hloubky. Jednotlivci, kteří dosáhli certifikace ISTQB® základní úrovně, se mohou také zajímat o pokročilé úrovně (analytik testování, technický analytik testování a manažer testování) a následně o expertní úrovně (management testování nebo zlepšování testovacího procesu). Každý, kdo chce rozvíjet dovednosti pro testování v agilním prostředí, může zvážit certifikace Agile Technical Tester nebo Agile Test Leadership at Scale. ISTQB® také nabízí v rámci kariérní cesty označované Specialist detailní pohled do oblastí, ve kterých jsou vyžadovány specifické přístupy k testování a specifické testovací činnosti (např. automatizace testů, testování AI, testování založené na modelu, testování mobilních aplikací) nebo které souvisí s konkrétními testovacími oblastmi (např. výkonnostní testování, testování použitelnosti, akceptační testování, bezpečnostní testování), případně kde je nezbytné mít speciální průmyslovou know-how (např. automobilový průmysl nebo hry). Aktuální informace o certifikacích testerů podle schématu ISTQB® lze nalézt na stránkách www.castb.org nebo www.istqb.org.

0.4 Profesní cíle

V této části je uvedeno 14 profesních cílů, které by měl splňovat každý, kdo dosáhl certifikace základní úrovně. Certifikovaný tester základní úrovně dokáže:

FL-BO1	Chápat, co je testování a proč je prospěšné.
FL-BO2	Chápat základní koncepty testování softwaru.
FL-BO3	Určit přístup k testování a činnosti, které mají být provedeny v závislosti na kontextu.

FL-BO4	Posoudit a zlepšit kvalitu dokumentace.
FL-BO5	Zvýšit efektivitu a účinnost testování.
FL-BO6	Sladit proces testování s životním cyklem vývoje softwaru.
FL-BO7	Rozumět principům managementu testování.
FL-BO8	Psát jasné a srozumitelné reporty o defektu a umět je komunikovat.
FL-BO9	Chápat faktory, které ovlivňují priority a pracnost činností souvisejících s testováním.
FL-BO10	Pracovat jako součást multifunkčního týmu.
FL-BO11	Rozumět rizikům a přínosům automatizace testů.
FL-BO12	Identifikovat nezbytné dovednosti potřebné pro testování.
FL-BO13	Chápat dopad rizik na testování.
FL-BO14	Efektivně reportovat o postupu a kvalitě testování.

0.5 Přezkoumatelné studijní cíle a kognitivní úrovně znalostí

Studijní cíle jsou odvozeny od profesních cílů a jsou použity k vytváření certifikačních zkoušek. Obecně lze říct, že veškerý obsah kapitol 1–6 v těchto osnovách je přezkoumatelný na úrovni K1. To znamená, že kandidát může být požádán, aby určil, zapamatoval si nebo si vybavil klíčové slovo nebo koncept z některé z šesti kapitol těchto osnov. Úrovně znalostí konkrétních studijních cílů jsou uvedeny na začátku každé kapitoly a jsou klasifikovány takto:

- K1: Zapamatovat si
- K2: Pochopit
- K3: Použít

Na začátku každé kapitoly (těsně pod názvem kapitoly) je uveden seznam klíčových slov. Všechna uvedená klíčová slova je nutné si zapamatovat (K1), a to i v případě, že nejsou výslovně uvedena ve studijních cílech. Více informací o klasifikaci úrovní znalostí včetně příkladů lze nalézt v příloze A.

0.6 Certifikační zkouška základní úrovně

Zkouška pro získání certifikátu základní úrovně je založena na těchto učebních osnovách a odpovědi na jednotlivé otázky při certifikační zkoušce mohou vyžadovat znalosti z různých kapitol. Všechny části těchto osnov mohou být předmětem certifikační zkoušky (kromě úvodu a příloh). Normy, knihy a jiné osnovy ISTQB® jsou uvedeny jako odkazy (viz kapitola 7), ale jejich obsah není součástí zkoušky s výjimkou toho, co je z nich přímo zahrnuto v těchto osnovách. Více informací lze nalézt v dokumentu Struktury a pravidla zkoušek základní úrovně v anglickém jazyce dostupné jako Foundation Level Examination Structures and Rules).

0.7 Akreditace

Členský výbor ISTQB® může akreditovat poskytovatele školení, jehož studijní materiály odpovídají těmto učebním osnovám. Pokyny pro akreditaci by měli poskytovatelé školení obdržet od daného členského výboru nebo orgánu, který akreditaci provádí. Akreditovaný kurz je následně uznán za vyhovující těmto osnovám. Do takového kurzu je možné i přímo začlenit certifikační zkoušku ISTQB®. Pokyny pro akreditaci pro tyto učební osnovy se řídí obecnými akreditační pokyny zveřejněnými příslušnou pracovní skupinou ISTQB® (v anglickém jazyce dostupné jako Accreditation Guidelines).

0.8 Využití norem

V těchto učebních osnovách jsou uvedeny odkazy na různé normy (např. normy IEEE nebo ISO). Informace v těchto normách mohou být citovány přímo (např. norma ISO 25010 při definici kvalitativních charakteristik) nebo poskytují případnému čtenáři zdroj dalších informací. Samotné normy však nejsou předmětem zkoušky. Více informací lze nalézt v kapitole 7.

0.9 Zajištění aktuálnosti

Softwarový průmysl se dynamicky mění. Aby bylo možné se s těmito změnami vypořádat a poskytnout všem zainteresovaným stranám přístup k relevantním a aktuálním informacím, vytvořily pracovní skupiny ISTQB® na webových stránkách www.istqb.org odkazy, které vedou na podpůrnou dokumentaci a změny norem. Znalost těchto informací není při zkoušce základní úrovně vyžadována.

0.10 Úroveň detailu

Úroveň detailu v těchto osnovách umožňuje provádět kurzy a zkoušky v mezinárodně srovnatelném rozsahu. Proto se osnovy skládají z:

- obecných cílů výuky popisujících záměr základní úrovně,
- seznamu pojmů (klíčových slov), které si studenti musí být schopni vybavit,
- studijních cílů pro každou oblast znalostí popisující výsledek učení, kterého má být dosaženo,
- popisu klíčových konceptů včetně odkazů na uznávané zdroje.

Obsahem osnov není vyčerpávající popis všech znalostí v oblasti testování softwaru, ale odráží úroveň detailu, která má být pokryta v rámci vzdělávacích kurzů pro základní úroveň. Obsah se zaměřuje na koncepci a techniky testování, které lze použít pro všechny softwarové projekty nezávisle na zvoleném SDLC (software development lifecycle, životní cyklus vývoje softwaru).

0.11 Uspořádání osnov

Součástí učebních osnov je celkem šest kapitol, jejichž obsah může být součástí zkoušky. Nejvyšší úroveň nadpisu pro každou kapitolu uvádí celkovou dobu výuky pro danou kapitolu, časování pro nižší úrovně kapitoly není definováno. Celkem je tak u akreditovaných vzdělávacích kurzů vyžadován rozsah nejméně 1135 minut (18 hodin a 55 minut) rozdělených do šesti kapitol a to takto:

- Kapitola 1: Základy testování (180 minut)
  - Studenti se seznámí se základními principy související s testováním, důvody, proč je testování vyžadováno, a s popisem cílů testování.
  - Studenti porozumí pojmům proces testování, hlavní testovací činnosti a testware.
  - Studenti se seznámí se základními dovednostmi nezbytnými pro testování.
- Kapitola 2: Testování v průběhu životního cyklu vývoje softwaru (130 minut)
  - Studenti porozumí tomu, jak je testování začleněno do různých metodik vývoje.
  - Studenti se seznámí s koncepty přístupů iniciovaných testy a s metodikami DevOps.
  - Studenti se seznámí s různými úrovněmi testů, typy testů a testováním údržby.
- Kapitola 3: Statické testování (80 minut)
  - Studenti se seznámí se základy statického testování, s procesem revizí a s principy předávání zpětné vazby.
- Kapitola 4: Testovací analýza a návrh testů (390 minut)
  - Studenti se naučí aplikovat techniky černé skříňky, bílé skříňky a techniky testování založené na zkušenostech při odvozování testovacích případů z různých softwarových pracovních produktů.
  - Studenti se seznámí s přístupem k testování založeným na spolupráci.
- Kapitola 5: Management testování (335 minut)
  - Studenti se naučí, jak obecně plánovat testy a jak odhadnout pracnost při testování.
  - Studenti se seznámí s tím, jak mohou rizika ovlivnit testování.
  - Studenti se naučí monitorovat a řídit testovací aktivity.
  - Studenti se seznámí s tím, jak konfigurační management podporuje testování.
  - Studenti se naučí reportovat defekty jasným a srozumitelným způsobem.

- Kapitola 6: Testovací nástroje (20 minut)
  - Studenti se naučí klasifikovat nástroje a porozumět rizikům a přínosům automatizace testů.

0.12 Použité zkratky

Následující tabulka uvádí seznam zkratek použitých v těchto osnovách, zároveň je význam zkratky uveden při jejím prvním výskytu. Definici některých zkratek lze také nalézt ve slovníku ISTQB® na adrese glossary.istqb.org.

Zkratka	Význam	Český význam
ISTQB®	International Software Qualifications Board	Mezinárodní výbor pro testování softwaru
CTFL	Certified Tester Foundation Level	Certifikovaný tester základní úrovně
SDLC	software development lifecycle	životní cyklus vývoje softwaru
QA	quality assurance	zajištění kvality
QC	quality control	kontrola kvality
ATDD	acceptance test driven development	vývoj řízený akceptačními testy
BDD	behavior driven development	vývoj řízený chováním
DDD	domain-driven design	návrh řízený doménou
XP	extreme programming	Extrémní programování
FDD	feature driven development	vývoj řízený užitnými vlastnostmi
TDD	test driven development	vývoj řízený testy
CI	continuous integration	průběžná integrace
CD	continuous delivery	průběžné dodávání
EP	equivalence partitioning	rozdělení tříd ekvivalence
BVA	boundary value analysis	analýza hraničních hodnot
API	application programming interface	aplikační programové rozhraní
E2E	end-to-end	nepřekládá se, významově jde o testování od začátku do konce
DDP	defect detection percentage	procento zjištěných defektů
KPI	key performance indicator	klíčový výkonnostní ukazatel

1 Základy testování – 180 minut

Klíčová slova

cíl testování, defekt, dokončení testování, chyba, implementace testování, kořenová příčina, kvalita, ladění, monitoring testování, návrh testů, ověřování, plánování testování, pokrytí, provedení testů, řízení testování, selhání, testovací analýza, testovací báze, testovací data, testovací podmínka, testovací procedura, testovací případ, testování, testovaný objekt, testware, validace, výsledek testu, zajištění kvality

Studijní cíle

1.1 Co je to testování?

FL-1.1.1 (K1) Identifikovat typické cíle testování.
FL-1.1.2 (K2) Odlišit testování od ladění.

1.2 Proč je testování nezbytné?

FL-1.2.1 (K2) Ilustrovat na příkladech, proč je testování nezbytné.
FL-1.2.2 (K1) Vybavit si vztah mezi testováním a zajištěním kvality.
FL-1.2.3 (K2) Rozlišit mezi kořenovou příčinou, chybou, defektem a selháním.

1.3 Principy testování

FL-1.3.1 (K2) Vysvětlit sedm principů testování.

1.4 Testovací činnosti, testware a role v testování

FL-1.4.1 (K2) Shrnout různé testovací činnosti a úkoly.
FL-1.4.2 (K2) Vysvětlit vliv kontextu na proces testování.
FL-1.4.3 (K2) Rozlišit mezi různými druhy testwaru podporujícími testovací činnosti.
FL-1.4.4 (K2) Vysvětlit hodnotu udržování trasovatelnosti.
FL-1.4.5 (K2) Porovnat různé role v testování.

1.5 Základní dovednosti a osvědčené postupy při testování

FL-1.5.1 (K2) Uvést příklady obecných dovedností požadovaných pro testování.
FL-1.5.2 (K1) Vybavit si výhody týmového přístupu.
FL-1.5.3 (K2) Uvědomit si výhody a nevýhody nezávislosti testování.

1.1 Co je to testování?

Softwarové systémy jsou nedílnou součástí našeho každodenního života. Většina lidí má zkušenosti se softwarem, který fungoval jinak, než očekávali. Software, který nefunguje správně, může vést k mnoha problémům, např. ztrátě peněz, času nebo obchodní pověsti a v extrémních případech dokonce ke zranění nebo smrti. Testování softwaru hodnotí kvalitu softwaru a pomáhá snižovat riziko selhání softwaru v provozu.

Testování softwaru je sada činností, jejichž cílem je odhalování defektů a vyhodnocování kvality softwarových artefaktů, které se při testování označují jako testované objekty. Běžná (a bohužel mylná) představa o testování je, že se jedná pouze o provádění testů (tj. spuštění softwaru a kontrola výsledků testů). Testování softwaru ale zahrnuje i další činnosti a musí být v souladu s životním cyklem vývoje softwaru (viz kapitola 2).

Další běžnou mylnou představou o testování je, že testování se zaměřuje výhradně na ověření testovaného objektu. Testování ale zahrnuje nejen ověření neboli verifikaci (tj. kontrolu, zda systém splňuje specifikované požadavky), ale také validaci (tedy kontrolu, zda systém splňuje potřeby uživatelů a ostatních zainteresovaných stran v jejich provozním prostředí).

Testování může být dynamické nebo statické. Dynamické testování vyžaduje spuštění testovaného softwaru, zatímco statické testování spuštění nevyžaduje. Statické testování zahrnuje revize (viz kapitola 3) a statickou analýzu. Dynamické testování používá různé techniky testování a přístupy k testování s cílem definovat testovací případy (viz kapitola 4).

Testování není jen technická činnost. Rovněž je třeba jej řádně plánovat, řídit, odhadovat, monitorovat a kontrolovat (viz kapitola 5).

Testeři sice používají při testování nástroje (viz kapitola 6), ale je důležité si uvědomit, že testování je hlavně intelektuální činnost. Při ní je potřebné, aby testeři měli specializované znalosti, používali analytické dovednosti a aplikovali kritické a systémové myšlení (Myers 2011, Roman 2018).

Další informace o konceptech testování softwaru lze nalézt v normě ISO/IEC/IEEE 29119-1.

1.1.1 Typické cíle testování

Typické cíle testování jsou:

- ohodnocení pracovních produktů jako jsou požadavky, uživatelské scénáře, návrhy a kód,
- vyvolání selhání a nalezení defektů,
- zajištění požadovaného pokrytí testovaného objektu,
- snížení úrovně rizika plynoucí z nedostatečné kvality softwaru,
- ověření, zda byly splněny stanovené požadavky,
- ověření, že testovaný objekt vyhovuje smluvním, právním a regulatorním požadavkům,
- poskytnutí informací zainteresovaným stranám tak, aby mohly přijmout kvalifikovaná rozhodnutí,
- vytvoření důvěry v danou úroveň kvality testovaného objektu,
- kontrola, zda je testovaný objekt kompletní a funguje dle očekávání všech zainteresovaných stran.

Cíle testování se mohou lišit v závislosti na kontextu, který zahrnuje faktory jako je charakteristika testovaného pracovního produktu, úroveň testování, rizika, zvolený životní cyklus vývoje softwaru (SDLC). Kromě toho ale závisí také na byznysovém kontextu s faktory jako struktura společnosti, konkurenční hlediska nebo doba uvedení na trh (time to market).

1.1.2 Testování a ladění

Testování a ladění (debugging) jsou samostatné činnosti. Testování může vyvolat selhání, která jsou způsobena defekty v softwaru (dynamické testování), nebo může přímo odhalit defekty v testovaném objektu (statické testování).

Zatímco dynamické testování (viz kapitola 4) se snaží vyvolat selhání, ladění se zabývá hledáním příčin takových selhání (tj. defektů), jejich analýzou a jejich odstraněním. Typický proces ladění v tomto případě zahrnuje:

- reprodukování selhání,
- diagnostika (nalezení kořenové příčiny),
- oprava příčiny.

Následné tzv. konfirmační testování ověřuje, zda došlo v důsledku těchto oprav k vyřešení problému. Provádí jej ideálně stejná osoba, která provedla prvotní test. Kromě toho lze také provést následné tzv. regresní testování s cílem ověřit, zda opravy nezpůsobují selhání v jiných částech testovaného objektu. Více informací o konfirmačním a regresním testování viz kapitola 2.2.3.

Statické testování přímo identifikuje defekt, ladění se pak zabývá jeho odstraněním. Obvykle při něm není nutno provádět reprodukování a diagnostiku, protože statické testování nachází přímo defekty a nemůže způsobit selhání (viz kapitola 3).

1.2 Proč je testování nezbytné?

Testování jako jeden ze způsobů řízení kvality pomáhá při dosažení dohodnutých cílů v rámci stanoveného rozsahu, času, kvality a rozpočtových omezení. K úspěchu projektu nepřispívají nutně pouze činnosti zajištěné testovacím týmem – kdokoliv ze zainteresovaných stran může využít své testovací dovednosti k tomu, aby se projekt přiblížil ke svému úspěšnému cíli. Testování komponent, systémů a související dokumentace pomáhá identifikovat defekty v softwaru.

1.2.1 Jak testování přispívá k úspěchu

Testování poskytuje nákladově efektivní způsob detekce defektů. Tyto defekty lze odstranit (laděním, tedy aktivitou, která nespadá pod oblast testování), takže testování vlastně nepřímo přispívá ke kvalitnějším testovaným objektům.

Testování poskytuje prostředky pro přímé hodnocení kvality testovaného objektu v různých fázích SDLC. Tato hodnocení se používají jako součást širších aktivit managementu projektů a přispívají tak k rozhodnutí o přechodu do další fáze SDLC, například k rozhodnutí o vydání. Testování tak vlastně poskytuje uživatelům nepřímou reprezentaci celého vývojového projektu.

Testeři jsou zárukou toho, že jejich pochopení potřeb uživatelů je zohledněno v průběhu celého životního cyklu vývoje. Alternativou k využití testerů je zapojení reprezentativní skupiny uživatelů do projektu, což ale obvykle není možné kvůli vysokým nákladům a nedostatečné dostupnosti vhodných uživatelů.

Vedle hodnocení kvality testovaného objektu může být testování softwaru vyžadováno také z důvodu splnění smluvních nebo zákonných požadavků či regulatorních norem.

1.2.2 Testování a zajištění kvality

Lidé často zaměňují pojmy testování a zajištění kvality (QA – quality assurance), nicméně není to totéž.

Testování je forma řízení kvality (QC – quality control). QC je nápravný přístup zaměřený na produkt a zaměřuje se na činnosti podporující dosažení odpovídající úrovně kvality. Testování je sice hlavní formou QC, nicméně není jedinou. Lze využít i jiné formální metody (např. ověření modelu nebo důkazy správnosti), případně simulace nebo prototypování.

QA je preventivní přístup orientovaný na procesy, který se zaměřuje na jejich zavádění a zlepšování. Je založen na předpokladu, že pokud je dodržován správný proces, pak je jeho výsledkem správný produkt. QA se vztahuje na procesy vývoje i testování a je odpovědností všech osob v projektu.

Výsledky testů jsou využívány v QA i v QC. V QC se používají k opravě defektů, zatímco v QA poskytují zpětnou vazbu o tom, jak dobře fungují procesy vývoje a testování.

1.2.3 Chyby, defekty, selhání a kořenové příčiny

Lidé dělají chyby (omyly), v jejichž důsledku vznikají defekty (vady, bugy), které mohou vést k selhání. Lidské chyby vznikají z různých důvodů jako jsou časová tíseň nebo složitost pracovních produktů, procesů,

infrastruktury a interakcí. Chyby mohou také vzniknout jednoduše proto, že jsou lidé unavení nebo jim chybí odpovídající školení.

Defekty lze nalézt v dokumentaci (např. ve specifikaci požadavků nebo testovacím skriptu), ve zdrojovém kódu nebo v jakémkoliv podpůrném artefaktu, např. v souboru sestavení (buildu). Pokud nejsou zjištěny defekty artefaktů vzniklých v dřívějších fázích SDLC, vedou často ke vzniku defektních artefaktů v jeho navazujících fázích.

Pokud je spuštěn defekt v kódu, systém nemusí dělat to, co by měl dělat nebo může udělat něco, co by neměl. To může (ale nemusí) způsobit selhání. Některé defekty mohou vést při spuštění k selhání vždy, jiné pouze za určitých okolností, některé dokonce nemusí selhání způsobit nikdy.

Chyby a defekty nejsou jedinou příčinou selhání. Selhání mohou být způsobena také podmínkami prostředí, příkladem může být defekt u firmwaru způsobený zářením nebo působením elektromagnetického pole.

Kořenová příčina je primárním důvodem vzniku problému (např. situace, která vede k chybě). Kořenové příčiny lze identifikovat pomocí jejich analýzy, která se obvykle provádí po výskytu selhání nebo identifikaci defektu. Předpokládá se, že řešením kořenové příčiny (nejlépe jejím odstraněním) lze zabránit podobným selháním nebo defektům (popř. alespoň snížit jejich četnost).

1.3 Principy testování

Během posledních let byla zdokumentována řada principů, které poskytují obecný návod aplikovatelný pro všechny druhy testování. Tyto učební osnovy popisují sedm takových principů.

1. Testování prokazuje přítomnost defektů, nikoli jejich nepřítomnost. Testování může prokázat, že defekty jsou v testovaném objektu přítomny, ale nemůže prokázat, že v něm žádné nejsou (Buxton 1970). Testování snižuje pravděpodobnost, že v testovaném objektu zůstaly neodhalené defekty, nicméně pokud nejsou žádné defekty nalezeny, není tím prokázána jeho správnost.

2. Kompletní testování není možné. Testování všeho (např. všech kombinací vstupů) není možné s výjimkou triviálních případů (Manna 1978). Místo snahy o kompletní testování je lepší se zaměřit na vhodné techniky testování (viz kapitola 4), stanovení priorit testovacích případů (viz kapitola 5.1.5) a testování založené na rizicích (viz kapitola 5.2).

3. Včasné testování šetří čas a peníze. Defekty, které jsou odstraněny v rané fázi procesu, nezpůsobí následně jiné defekty v odvozených (následných) pracovních produktech. Náklady na kvalitu budou navíc sníženy, protože později v SDLC dojde k menšímu počtu selhání (Boehm 1981). Pro včasné odhalení defektů by mělo být co nejdříve zahájeno jak statické testování (viz kapitola 3), tak dynamické testování (viz kapitola 4).

4. Shlukování defektů. Většinu zjištěných defektů obsahuje obvykle malé množství systémových komponent, případně je malé množství systémových komponent zodpovědných za většinu provozních selhání (Enders 1975). Tento jev je ilustrací tzv. Paretova pravidla. Předpokládané a skutečně pozorované shluky defektů během testování nebo v provozu jsou významným vstupem pro testování založené na rizicích (viz kapitola 5.2).

5. Testy se opotřebovávají. Pokud se stejné testy opakují neustále dokola, stávají se stále méně účinnými a nedochází k odhalování nových defektů (Beizer 1990). Abychom se vypořádali s důsledky tohoto principu, je nutné měnit stávající testy a testovací data, příp. je potřeba vytvořit testy nové. V některých případech má však opakování stejných testů význam, např. při automatizovaném regresním testování (viz kapitola 2.2.3).

6. Testování je závislé na kontextu. Neexistuje jediný univerzálně použitelný přístup k testování, testování se provádí odlišně v různých kontextech (Kaner 2011).

7. Nepřítomnost defektů je klam. Je mylnou představou očekávat, že pouhé ověřování (verifikace) softwaru zajistí jeho úspěch. I při důkladném testování všech specifikovaných požadavků a odstranění všech zjištěných defektů může vzniknout systém, který nesplňuje potřeby a očekávání uživatelů, který nepomáhá dosahovat byznysových cílů zákazníka, a který je horší ve srovnání s jinými konkurenčními systémy. Kromě ověřování by měla být provedena také validace (Boehm 1981).

1.4 Testovací činnosti, testware a role v testování

Testování je sice závislé na kontextu, nicméně existují testovací činnosti, bez nichž je méně pravděpodobné, že bude dosaženo vytyčených cílů. Skupiny takových činností nazýváme proces testování. Proces testování může být přizpůsoben dané situaci na základě různých faktorů. O tom, které testovací činnosti jsou zahrnuty do procesu testování, jak jsou implementovány a kdy k nim dojde, se obvykle rozhoduje v rámci plánování testování pro konkrétní situaci (viz kapitola 5.1).

Následující text popisuje obecné aspekty tohoto procesu z hlediska testovacích činností a úkolů, vlivu kontextu, testwaru, trasovatelnosti mezi testovací bází a testwarem a také rolí v testování.

Další informace o procesech testování lze nalézt v normě ISO/IEC/IEEE 29119-2.

1.4.1 Testovací činnosti a úkoly

Proces testování se obvykle skládá z hlavních skupin činností (viz dále), které je obvykle nutno přizpůsobit systému a projektu. I když se mnohé z těchto hlavních skupin činností mohou jevit jako logicky seřazené za sebou, jsou často prováděny iterativně nebo paralelně.

Plánování testování se skládá z definování cílů testování a následného výběru takového přístupu, který nejlépe dosahuje definovaných cílů v rámci omezení daných celkovým kontextem. Plánování testování je dále vysvětleno v kapitole 5.1.

Monitoring a řízení testování. Monitoring testování zahrnuje průběžnou kontrolu všech testovacích činností a porovnávání skutečného stavu proti plánu, řízení testování se zabývá přijetím opatření nezbytných pro dosažení cílů testování. Monitoring a řízení testování jsou dále vysvětleny v kapitole 5.3.

Testovací analýza zahrnuje analýzu testovací báze za účelem identifikace testovatelných užitných vlastností (features) a definování příslušných testovacích podmínek včetně stanovení jejich priorit a souvisejících rizik spolu s jejich úrovněmi (viz kapitola 5.2). Testovací báze a testované objekty jsou také hodnoceny za účelem posouzení jejich testovatelnosti a identifikace případných defektů v nich obsažených. Při testovací analýze jsou často používány techniky testování (viz kapitola 4). Testovací analýza určuje „Co se má testovat?“ ve smyslu měřitelných kritérií pokrytí.

Návrh testů zahrnuje rozpracování testovacích podmínek do testovacích případů a dalšího testwaru (např. testovacích listin). Často obsahuje také identifikaci položek pokrytí sloužících jako vodítko pro stanovení vstupů testovacích případů. Při této činnosti lze použít taktéž vhodné techniky testování (viz kapitola 4). Návrh testů také zahrnuje stanovení požadavků na testovací data, návrh testovacího prostředí a identifikaci další potřebné infrastruktury nebo nástrojů. Návrh testů dává odpověď na otázku „Jak testovat?“.

Implementace testování zahrnuje vytvoření nebo získání testwaru nezbytného pro provedení testů (např. testovací data). Testovací případy mohou být uspořádány do testovacích procedur a jsou často seskupeny do testovacích sad. Mimo jiné jsou v rámci této činnosti vytvářeny manuální a automatizované testovací skripty. Z důvodu efektivního provedení testů (viz kapitola 5.1.5) jsou testovací procedury prioritizovány a uspořádány v harmonogramu provedení testů. Součástí činností implementace testování je sestavení testovacího prostředí a ověření jeho správného nastavení.

Provedení testů zahrnuje spuštění testů v souladu s harmonogramem provedení testů (jako dílčích běhů testů, test runs). Provedení testů může být manuální nebo automatizované a může mít mnoho forem včetně tzv. průběžného nebo párového testování. V rámci této činnosti jsou porovnány skutečné výsledky testů s očekávanými a zaznamenány jejich výsledky. Anomálie jsou analyzovány s cílem identifikace jejich pravděpodobné příčiny. Tato analýza umožňuje reportovat anomálie na základě pozorovaných selhání (viz kapitola 5.5).

K činnostem **dokončení testování** obvykle dochází při dosažení milníků projektu (např. vydání, konec iterace, dokončení úrovně testování). Pro všechny nevyřešené defekty jsou vytvořeny změnové požadavky nebo položky produktového backlogu. Jakýkoli testware, který může být v budoucnu užitečný, je identifikován a archivován, případně je předán příslušným týmům. Testovací prostředí se uvede do dohodnutého stavu (např.

je vypnuto). Testovací činnosti jsou analyzovány s cílem identifikace ponaučení (lessons learned) a vylepšení pro případné budoucí iterace, vydání nebo i jiné projekty (viz kapitola 2.1.6). Je vytvořen souhrnný report) z testování, který je komunikován příslušným zainteresovaným stranám.

1.4.2 Proces testování v souvislostech

Testování neprobíhá izolovaně. Testovací činnosti jsou nedílnou součástí procesů vývoje prováděných v rámci organizace. Finálním cílem testování je (mimo jiné) pomoci splnit byznysové potřeby zainteresovaných stran, které také proces testování financují. Způsob, jakým se testování provádí, bude proto záviset na řadě faktorů, například:

- zainteresované strany (jejich potřeby, očekávání, požadavky, ochota spolupracovat),
- členové týmu (jejich dovednosti, znalosti, úroveň zkušeností, dostupnost, potřeby školení),
- obor podnikání (kritičnost testovaného objektu, identifikovaná rizika, potřeby trhu, konkrétní právní předpisy),
- technické faktory (typ softwaru, architektura produktu, použité technologie),
- omezení projektu (rozsah, čas, rozpočet, zdroje),
- organizační faktory (organizační struktura, existující politiky, používané postupy),
- životní cyklus vývoje softwaru (technické postupy, metody vývoje atd.),
- nástroje (dostupnost, použitelnost, shoda s předpisy).

Tyto faktory budou mít dopad na mnoho aspektů souvisejících s testováním včetně strategie testování, použitých technik testování, stupně automatizace testů, požadované úrovně pokrytí, úrovně podrobnosti dokumentace testování, reportování atd.

1.4.3 Testware

Testware je definován jako výstupní pracovní produkt z testovacích činností popsaných v kapitole 1.4.1. V tom, jak různé organizace vytvářejí, pojmenovávají, organizují a spravují takové pracovní produkty, existují nicméně značné rozdíly. Pro podporu konzistence a integrity testwaru je důležité zajištění dobrého konfiguračního managementu (viz kapitola 5.4).

Následující seznam uvádí některé (ale ne všechny) takové typické výstupní pracovní produkty (testware):

- **Pracovní produkty z plánování testování**: plán testování, harmonogram testování, registr rizik a vstupní a výstupní kritéria (viz kapitola 5.1). Registr rizik je seznam rizik spolu s jejich pravděpodobností, dopadem a informacemi o jejich možném zmírnění (risk mitigation), viz kapitola 5.2. Harmonogram testování, registr rizik, vstupní a výstupní kritéria jsou často součást plánu testování.
- **Pracovní produkty z monitoringu a řízení testování**: reporty o postupu prací při testování (viz kapitola 5.3.2), pokyny a nezbytná nápravná opatření (viz kapitola 5.3) a informace o rizicích (viz kapitola 5.2).
- **Pracovní produkty z testovací analýzy**: (prioritizované) testovací podmínky (např. akceptační kritéria, viz kapitola 4.5.2) a reporty o defektech v testovací bázi (pokud už nebyly opraveny).
- **Pracovní produkty z návrhu testů**: (prioritizované) testovací případy, testovací listiny, položky pokrytí, požadavky na testovací data a požadavky na testovací prostředí.
- **Pracovní produkty z implementace testování**: testovací procedury, automatizované testovací skripty, testovací sady, testovací data, harmonogram provedení testů a prvky testovacího prostředí. Mezi příklady položek testovacího prostředí patří stuby, ovladače, simulátory a virtualizace služeb.
- **Pracovní produkty z provedení testů**: protokoly z testování (test logs) a reporty o defektech (viz kapitola 5.5).
  Pracovní produkty z dokončení testování: souhrnný report z testování (viz kapitola 5.3.2), akční body pro zlepšení budoucích projektů nebo iterací, zdokumentované ponaučení (lessons learned) a změnové požadavky (např. jako položky produktového backlogu).

1.4.4 Trasovatelnost mezi testovací bází a pracovními produkty z testování

Aby bylo možné efektivně monitorovat a řídit testování, je důležité vytvořit a následně udržovat v průběhu celého procesu testování trasovatelnost mezi každou položkou testovací báze, výstupními pracovními produkty

vztaženým k této položce (testwarem – např. testovací podmínky, rizika nebo testovací případy), výsledky testů a nalezenými defekty.

Přesná trasovatelnost umožňuje vyhodnocovat pokrytí, takže je velmi užitečné, pokud jsou v testovací bázi pro toto pokrytí definována jeho měřitelná kritéria. Kritéria pokrytí mohou představovat jeden z klíčových ukazatelů výkonnosti (KPI – key performance indicator) pro řízení testovacích činností, která ukazují na míru dosažení cílů testování (viz kapitola 1.1.1). Například:

- Trasovatelnost mezi testovacími případy a požadavky pomůže ověřit, že požadavky jsou pokryty testovacími případy.
- Trasovatelnost mezi výsledky testů a riziky lze použít k vyhodnocení úrovně zbytkového rizika u testovaného objektu.

Kromě vyhodnocení pokrytí umožňuje dobrá trasovatelnost určit dopad změn, usnadňuje provádění auditů testování a pomáhá plnit kritéria zásad správného řízení v IT. Dobrá trasovatelnost rovněž zlepšuje srozumitelnost reportů o postupu prací při testování a souhrnných reportů z testování tím, že ukazuje stav položek testovací báze. To může rovněž přispět ke srozumitelné komunikaci se zainteresovanými stranami o technických aspektech testování. Trasovatelnost poskytuje informace pro posouzení kvality produktu, způsobilosti procesů a postupu prací v projektu vzhledem k byznysovým cílům.

1.4.5 Role v testování

V těchto učebních osnovách jsou definovány dvě hlavní role při testování: role manažera testování a role testera. Činnosti a úkoly vykonávané těmito dvěma rolemi závisí na kontextu projektu a produktu, na dovednostech lidí v daných rolích a na dané organizaci.

Manažer testování má celkovou odpovědnost za proces testování, testovací tým a úspěšné řízení testovacích činností. Zaměřuje se především na činnosti plánování testování, monitoringu a řízení testování a dokončení testování. Způsob, jakým se tato role vykonává, se liší v závislosti na životním cyklu vývoje softwaru, například v agilním vývoji jsou některé úkoly manažera testování vykonávány celým agilním týmem. Úkoly, které zahrnují více týmů nebo celou organizaci, mohou být prováděny manažery testování mimo vývojový tým.

Tester přebírá celkovou odpovědnost za inženýrský (technický) aspekt testování. Zaměřuje se především na činnosti testovací analýzy, návrhu testů, implementace testování a provedení testů.

Obě role mohou být vykonávány v různých obdobích různými lidmi, například role manažera testování může být vykonávaná vedoucím týmu, samotným manažerem testování nebo manažerem vývoje. Je také možné, aby jedna osoba převzala roli testera i roli manažera testování současně.

1.5 Základní dovednosti a osvědčené postupy při testování

Dovednost je schopnost dělat něco dobře a vychází z něčích znalostí, praxe a schopností. Aby mohli testeři dobře vykonávat svou práci, potřebují mít určité základní dovednosti, mimo jiné by měli být dobrými týmovými hráči a měli by být schopni pracovat v různých úrovních nezávislosti testování.

1.5.1 Obecné dovednosti potřebné pro testování

Následující dovednosti jsou sice obecné, ale pro testery obzvláště důležité:

- Znalost testování (s cílem zvyšování efektivity testování, např. pomocí technik testování).
- Důkladnost, pečlivost, zvídavost, důraz na detail, schopnost pracovat metodicky (s cílem identifikace defektů, a to zejména těch, které lze jen obtížně najít).
- Dobré komunikační dovednosti, aktivní naslouchání, schopnost být týmovým hráčem (s cílem efektivně komunikovat se všemi zainteresovanými stranami, předávat informace ostatním, být pochopen, reportovat defekty a diskutovat o nich).
- Analytické myšlení, kritické myšlení, kreativita (s cílem zvyšování efektivity testování).
- Technické znalosti (s cílem zvyšování efektivity testování, např. použitím vhodných testovacích nástrojů).

- Znalost domény (s cílem mít schopnost porozumět koncovým uživatelům nebo obchodním zástupcům a komunikovat s nimi).

Běžnou lidskou vlastnosti je obviňování nositelů špatných zpráv a testeři jsou často nositeli takových špatných zpráv. Z tohoto důvodu jsou pro testery klíčové komunikační dovednosti – sdělování výsledků testů může být totiž vnímáno jako kritika produktu a jeho autora.

Termín konfirmační zkreslení (confirmation bias) popisuje tendenci člověka obtížně přijímat informace, které jsou v rozporu s jeho stávajícími názory. Jedná se o psychologický faktor, v jehož důsledku mohou někteří lidé vnímat testování jako destruktivní aktivitu, přestože významně přispívá k úspěchu projektu a kvalitě produktu. Při snaze snížit tato vnímání by informace o defektech a selháních měly být komunikovány konstruktivně a bez emocí.

1.5.2 Týmový přístup

Jednou z důležitých dovedností testerů je schopnost efektivně pracovat v týmu a pozitivně přispívat k týmovým cílům. Tzv. týmový přístup (whole team approach, technika vycházející z extrémního programování, viz kapitola 2.1) je postaven na této dovednosti.

Při týmovém přístupu může každý člen týmu s potřebnými znalostmi a dovednostmi vykonávat jakýkoliv úkol a každý člen týmu je zodpovědný za kvalitu. Členové týmu sdílejí stejný pracovní prostor, protože společné umístění v jednom fyzickém nebo virtuálním prostoru usnadňuje vzájemnou komunikaci a interakci. Týmový přístup zvyšuje týmovou dynamiku, zlepšuje komunikaci a spolupráci v rámci týmu a vytváří synergii tím, že umožňuje využití různých dovedností v týmu ve prospěch projektu.

Pokud má být dosaženo požadované úrovně kvality, musí testeři úzce spolupracovat s ostatními členy týmu. To zahrnuje (mimo jiné) spolupráci se zástupci byznysu (kteří mohou např. pomoci s vytvořením vhodných akceptačních testů) nebo s vývojáři (např. s cílem dosahovat společné dohody o strategii testování a o přístupech k automatizaci testů). Testeři mohou také předávat znalost testování ostatním členům týmu a ovlivňovat vývoj produktu.

Týmový přístup nemusí být v kontextu projektu vždy vhodnou metodikou. Typickým příkladem je testování bezpečnostně kritických scénářů, kdy je nutné zachovat vysokou úroveň nezávislosti testování.

1.5.3 Nezávislost testování

Určitá míra nezávislosti činí testery efektivnějšími při hledání defektů, což je způsobeno rozdíly mezi kognitivním zkreslením autorů (vývojářů) a testerů (Salman 1995). Nezávislost však není náhrada za dobrou znalost systému – i vývojáři mohou efektivně najít mnoho defektů ve svém vlastním kódu.

Pracovní produkty mohou být testovány jejich autorem (nulová nezávislost), kolegy autora ze stejného týmu (malá nezávislost), testery mimo tým autora, ale v rámci organizace (vysoká nezávislost) nebo testery mimo organizaci (velmi vysoká nezávislost). U většiny projektů je obvykle nejlepší provádět testování s více úrovněmi nezávislosti (např. vývojáři provádějící testování komponent a integrační testování komponent, testovací tým provádějící systémové testování a systémové integrační testování a obchodní zástupci provádějící akceptační testování).

Hlavním přínosem nezávislosti testování je to, že nezávislí testeři dokáží pravděpodobněji rozpoznat různé druhy selhání a defektů ve srovnání s vývojáři, a to z důvodu jejich odlišného zázemí, technického nadhledu a předsudků. Nezávislí testeři mohou navíc ověřit, napadnout či vyvrátit předpoklady, které měly všechny zainteresované strany v době přípravy specifikace či implementace systému.

Existují však i některé nevýhody. Nezávislí testeři mohou být izolováni od vývojového týmu, což může vést k nedostatečné spolupráci, problémům v komunikaci nebo nepřátelskému vztahu s vývojovým týmem. Vývojáři mohou ztratit pocit zodpovědnosti za kvalitu. Nezávislí testeři mohou být úzkým místem ve vývojovém procesu nebo mohou být viněni za zpoždění při vydávání produktu.

2 Testování v průběhu životního cyklu vývoje softwaru – 130 minut

Klíčová slova

akceptační testování, funkcionální testování, integrační testování, integrační testování komponent, konfirmační testování, nefunkcionální testování, regresní testování, shift-left, systémové integrační testování, systémové testování, testování bílé skříňky, testování černé skříňky, testování komponent, testování údržby, typ testu, úroveň testování

Studijní cíle

2.1 Testování v kontextu životního cyklu vývoje softwaru

FL-2.1.1 (K2) Vysvětlit vliv zvoleného životního cyklu vývoje softwaru na testování.
FL-2.1.2 (K1) Vybavit si osvědčené postupy při testování aplikovatelné na všechny typy SDLC.
FL-2.1.3 (K1) Vybavit si příklady vývoje iniciovaného testy (test-first).
FL-2.1.4 (K2) Shrnout, jaký vliv na testování může mít DevOps.
FL-2.1.5 (K2) Vysvětlit přístup k testování shift-left.
FL-2.1.6 (K2) Vysvětlit, jak je možné využít retrospektivy jako mechanismu pro zlepšování procesů.

2.2 Úrovně testování a typy testů

FL-2.2.1 (K2) Rozlišit mezi různými úrovněmi testování.
FL-2.2.2 (K2) Rozlišit mezi různými typy testů.
FL-2.2.3 (K2) Odlišit konfirmační testování od regresního testování.

2.3 Testování údržby

FL-2.3.1 (K2) Shrnout testování údržby a jeho spouštěče.

2.1 Testování v kontextu životního cyklu vývoje softwaru

Model životního cyklu vývoje softwaru (SDLC) je zobecněný proces vývoje softwaru. Určuje, jak spolu logicky i chronologicky souvisí různé fáze vývoje a typy činností prováděných v rámci tohoto procesu. Mezi kategorie modelů SDLC patří sekvenční modely vývoje (např. vodopádový model, V-model), iterativní modely vývoje (např. spirálový model, prototypování) a inkrementální vývojové modely (např. Rational Unified Process).

Některé činnosti v rámci procesů vývoje softwaru lze také popsat podrobnějšími metodami vývoje softwaru a agilními postupy. Mezi takové metody patří vývoj řízený akceptačními testy (ATDD – acceptance test driven development), vývoj řízený chováním (BDD – behavior driven development), návrh řízený doménou (DDD – domain-driven design), extrémní programování (XP – extreme programming), vývoj řízený užitnými vlastnostmi (FDD – feature driven development), Kanban, Lean IT, Scrum a vývoj řízený testováním (TDD – test driven development).

2.1.1 Vliv životního cyklu vývoje softwaru na testování

Aby bylo testování úspěšné, musí být přizpůsobeno SDLC. Volba SDLC má dopad na:

- rozsah a načasování testovacích činností (např. úrovně testování a typy testů),
- úroveň detailu testovací dokumentace,
- volba technik testování a přístupu k testování,
- rozsah automatizace testů,
- role a zodpovědnosti testerů.

V sekvenčních modelech vývoje se testeři v počátečních fázích obvykle účastní revizí požadavků, testovací analýzy a návrhu testů. Jelikož je spustitelný kód obvykle vytvořen až v pozdějších fázích SDLC, nelze v těchto počátečních fázích používat techniky dynamického testování.

V některých iterativních a inkrementálních vývojových modelech se předpokládá, že výsledkem každé iterace je funkční prototyp nebo přírůstek produktu. To znamená, že v každé iteraci může být provedeno statické i dynamické testování, a to ve všech úrovních testování. Časté dodávání takových přírůstků vyžaduje rychlou zpětnou vazbu a rozsáhlé regresní testování.

Agilní vývoj softwaru (s využitím iterativních a inkrementálních modelů) předpokládá, že v průběhu projektu může dojít ke změně. Proto je v agilních projektech upřednostňována spíše stručnější dokumentace pracovních produktů, a naopak rozsáhlá automatizace testů, která usnadňuje regresní testování. Většina manuálních testů je často prováděna pomocí testovacích technik založených na zkušenostech (viz kapitola 4.4), u kterých není vyžadováno provedení rozsáhlé testovací analýzy a návrhu testů.

2.1.2 Životní cyklus vývoje softwaru a osvědčené testovací postupy

Mezi osvědčené testovací postupy, nezávisle na zvoleném modelu SDLC, patří:

- Ke každé vývojové činnosti existuje odpovídající testovací činnost, takže všechny vývojové činnosti podléhají řízení kvality.
- Existují různé úrovně testování (viz kapitola 2.2.1) a každá má své specifické (a někdy i odlišné) cíle. Tím je testování přiměřeně detailní a zároveň nedochází k nadbytečnostem.
- Testovací analýza a návrh testů pro danou úroveň začíná během odpovídající vývojové fáze SDLC, takže je splněn princip včasného testování (viz kapitola 1.3)
- V okamžiku, kdy je k dispozici pracovní verze (draft) libovolného pracovního produktu, jsou do jeho revize zapojeni testeři, což (jako forma včasného testování a včasné detekce defektů) podporuje přístup shift-left (viz kapitola 2.1.5).

2.1.3 Vývoj softwaru řízený testováním

TDD, ATDD a BDD jsou podobné přístupy k vývoji, kdy jsou testy definovány jako prostředek pro řízení vývoje. Každý z těchto přístupů je aplikací principu včasného testování (viz kapitola 1.3) a uplatňuje přístup shift-left (viz

kapitola 2.1.5) – testy jsou definovány před tvorbou samotného kódu. Všechny podporují iterativní model vývoje a jsou charakterizovány následovně:

Vývoj řízený testy (TDD)

- Psaní kódu je řízeno pomocí testovacích případů (místo rozsáhlého návrhu softwaru) (Beck 2003).
- Nejprve jsou sepsány testy a až následně kód a to tak, aby těmto testům vyhovoval. Následně mohou (ale nemusí) být testy a kód refaktorovány.

Vývoj řízený akceptačními testy (ATDD)

- Odvozuje testy z akceptačních kritérií jako součást procesu návrhu systému (Gärtner 2011).
- Daná část aplikace je vytvořena tak, aby vyhovovala testům napsaným před samotným vývojem.

Více informací o ATDD lze nalézt v kapitole 4.5.3.

Vývoj řízený chováním (BDD)

- Požadované chování aplikace je popsáno testovacími případy napsanými v jednoduché formě přirozeného jazyka srozumitelného pro zainteresované strany. Obvykle je využit formát Given/When/Then (Vstup/Podmínka/Akce) (Chelimsky 2010).
- Testovací případy jsou následně automaticky přeloženy do spustitelných testů.

Pro všechny výše uvedené přístupy mohou být testy automatizovány (ale nemusí) s cílem podpořit kvalitu při budoucích úpravách nebo refaktoringu.

2.1.4 DevOps a testování

DevOps je přístup k vývoji softwaru, který je postaven na synergii mezi odděleními vývoje (včetně testování) a provozu s cílem dosahovat definovaných společných cílů. DevOps vyžaduje změnu kultury v rámci organizace tak, aby se mezi těmito dvěma organizačními jednotkami překlenuly mezery a zároveň se k nim přistupovalo se stejnou důležitostí. DevOps podporuje autonomii týmu, rychlou zpětnou vazbu, integrované sady nástrojů a užití technických přístupů jako je průběžná integrace (CI – continuous integration) a průběžné dodávání (CD – continuous delivery). To umožňuje týmům vytvářet, testovat a vydávat kvalitní kód rychleji prostřednictvím definované DevOps pipeline (Kim 2016).

Z pohledu testování má DevOps tyto výhody:

- Poskytuje rychlou zpětnou vazbu o kvalitě (nového) kódu a o nepříznivém vlivu na dosavadní funkčnost systému/komponenty.
- Díky CI podporuje přístup shift-left při testování (viz kapitola 2.1.5) tím, že motivuje vývojáře k psaní kvalitního kódu podporovaného testováním komponent a statickou analýzou.
- Propaguje automatizované zpracování (např. CI/CD), které usnadňuje vytváření stabilních testovacích prostředí.
- Zvyšuje důraz na nefunkcionální charakteristiky kvality (např. výkon nebo spolehlivost).
- Automatizací prostřednictvím pipeline snižuje potřebu opakovaného manuálního testování.
- Minimalizuje riziko regrese díky rozsahu a míře automatizovaných regresních testů.

DevOps má také některá rizika a nevýhody:

- Musí být definovány a zavedeny DevOps pipeline.
- Musí být zavedeny a udržovány nástroje CI/CD.
- Automatizované testy vyžadují dodatečné investice a může být obtížné je vytvořit a udržovat.

Přestože DevOps předpokládá vyšší rozsah automatizovaného testování, nelze opomíjet ani manuální testování, a to zejména z pohledu koncového uživatele.

2.1.5 Přístup shift-left

Princip včasného testování (viz kapitola 1.3) je někdy označován jako shift-left (posun doleva ve smyslu časové osy), kdy je testování prováděno v dřívějších fázích SDLC. Shift-left doporučuje začít s testováním dříve (např.

nečekat na implementaci kódu nebo na integraci komponent), ale neznamená to, že by testování v pozdějších fázích mělo být zanedbáváno.

Existují některé osvědčené postupy, na kterých lze demonstrovat aplikaci tohoto přístupu:

- Revidování specifikací z pohledu testování, které nachází potenciální defekty jako jsou nejednoznačnosti, neúplnosti a nesrovnalosti.
- Psaní testovacích případů před napsáním kódu a spuštění kódu v sadě testovacího vybavení (test harness) během vývoje kódu.
- Používání CI anebo lépe CD, protože přichází s rychlou zpětnou vazbou a automatizovanými testy komponent, které jsou společně s kódem uloženy do repozitáře.
- Spouštění statické analýzy zdrojového kódu před dynamickým testováním nebo jako součást daného automatizovaného procesu.
- Provádění nefunkcionálních testů hned v úrovni testování komponent (pokud je to možné). Jedná se také o formu přístupu shift-left, jelikož obvykle jsou tyto typy testů prováděny v pozdějších fázích SDLC, kdy je k dispozici kompletní systém a odpovídající testovací prostředí.

Zavedení shift-left přístupu může v počátečních fázích znamenat riziko navýšení nákladů ve formě víceprací nebo nutnosti proškolení týmu, ale v pozdějších fázích naopak šetří pracnost a/nebo náklady.

Je důležité, aby byli všichni zástupci zainteresovaných stran přesvědčeni o užitečnosti tohoto konceptu a podporovali jej.

2.1.6 Retrospektivy a zlepšování procesů

Retrospektivy (známé také jako po-projektové schůzky nebo projektové retrospektivy) se často konají na konci projektu nebo iterace, při dosažení milníku nebo dle potřeby (ad-hoc). Načasování a organizace retrospektiv závisí na konkrétním modelu SDLC. Účastníci těchto schůzek (nejen testeři, ale také např. vývojáři, architekti, vlastníci produktů, byznysoví analytici) obvykle probírají:

- Co bylo úspěšné a mělo by být zachováno?
- Co se nepovedlo a dalo by se zlepšit?
- Jak implementovat návrhy na zlepšení a zajistit v budoucnu trvalou úspěšnost?

Výstupy z retrospektiv by měly být zaznamenány, obvykle jsou součástí souhrnného reportu z testování (viz kapitola 5.3.2). Retrospektivy jsou z hlediska nastavení procesu kontinuálního zlepšování kritické a je důležité, aby byla všechna doporučená opatření sledována (a dodržována).

Mezi typické výhody retrospektiv z pohledu testování patří:

- Zvyšování efektivity / účinnosti testů (např. z důvodu implementace návrhů na zlepšení procesů).
- Zvyšování kvality testwaru (např. společnou revizí testovacích procesů).
- Stmelování týmu a vzájemné učení (např. jako výsledek možnosti hlásit problémy a navrhovat zlepšení).
- Zlepšování kvality testovací báze (např. vyřešením nedostatků v rozsahu a kvalitě požadavků).
- Zlepšování spolupráce mezi vývojem a testováním (např. díky pravidelným revizím a optimalizaci vzájemné spolupráce).

2.2 Úrovně testování a typy testů

Úrovně testování jsou skupiny testovacích činností, které jsou organizovány a řízeny společně. Každá úroveň testování je instancí procesu testování tvořenou činnostmi prováděnými ve vztahu k softwaru v dané fázi vývoje, od jednotlivých komponent až po celé systémy, nebo případně i systémy systémů.

Úrovně testování souvisí s dalšími činnostmi v rámci životního cyklu vývoje softwaru. V sekvenčních modelech SDLC jsou úrovně testování často definovány tak, že výstupní kritéria jedné úrovně jsou součástí vstupních kritérií další úrovně. To ale nemusí platit v některých iterativních modelech, kde se úrovně testování mohou v čase překrývat a vývojové činnosti mohou překlenout více úrovní testování.

Typy testů jsou skupiny testovacích činností související se specifickými kvalitativními charakteristikami, kdy většinu těchto činností lze provádět v libovolné úrovni testování.

2.2.1 Úrovně testování

V těchto učebních osnovách je popsáno následujících pět úrovní testování:

- **Testování komponent** (označované také jako jednotkové testování, unit testing) se zaměřuje na testování izolovaných komponent. Pro jejich provádění je často potřebný speciální software, např. různé sady testovacího vybavení (test harness) nebo frameworky pro jednotkové testování. Testování komponent obvykle provádějí vývojáři ve svých vývojových prostředích.
- **Integrační testování komponent** (označované také jako integrační testování jednotek) se zaměřuje na testování rozhraní a interakcí mezi komponentami. Toto testování je silně závislé na zvolené integrační strategii (např. zdola-nahoru, shora-dolů nebo tzv. velký třesk).
- **Systémové testování** se zaměřuje na celkové chování a schopnosti celého systému nebo produktu, často včetně funkcionálního testování komplexních (end-to-end) úkolů a nefunkcionálního testování kvalitativních charakteristik. Některé z nich (např. použitelnost) je vhodnější testovat na úplném systému ve vhodném testovacím prostředí. Kromě jiného lze využívat simulace subsystémů. Systémové testování vychází ze specifikací systému a může být prováděno nezávislým testovacím týmem.
- **Systémové integrační testování** se zaměřuje na testování rozhraní mezi testovaným systémem a dalšími systémy nebo externími službami. Pro systémové integrační testování je potřebné mít vhodné testovací prostředí, pokud možno podobné provoznímu.
- **Akceptační testování** se zaměřuje na prokázání připravenosti systému k nasazení do produkčního prostředí a validuje, že systém splňuje byznysové potřeby uživatele. V ideálním případě by akceptační testování měli provádět koncoví uživatelé. Nejčastější formy akceptačního testování jsou: uživatelské akceptační testování (UAT), provozní akceptační testování, smluvní a regulatorní akceptační testování, alfa testování a beta testování.

Aby nedocházelo k překrývání činností při testování v jednotlivých úrovních, dílčí úrovně testování se odlišují specifickou definicí různých atributů. Mezi nejdůležitější atributy patří:

- testovaný objekt,
- cíle testování,
- testovací báze,
- defekty a selhání,
- přístup a zodpovědnosti.

2.2.2 Typy testů

Existuje mnoho typů testů, které lze na projektech použít. Tyto učební osnovy se zabývají čtyřmi typy testů.

Funkcionální testování ověřuje funkcionality, které by komponenta nebo systém měl vykonávat. Funkcionality představují to, „co“ by systém měl dělat. Hlavním cílem funkcionálního testování je kontrola funkcionální úplnosti, funkcionální správnosti a funkcionální vhodnosti.

Nefunkcionální testování vyhodnocuje nefunkcionální charakteristiky komponenty nebo systému a dává tak odpověď na otázku „jak dobře se systém chová“. Seznam nefunkcionálních charakteristik kvality softwaru lze nalézt v normě ISO/IEC 25010:

- výkonnostní efektivita,
- kompatibilita,
- použitelnost,
- bezporuchovost (spolehlivost),
- bezpečnost,
- udržovatelnost,
- přenositelnost.

Mnoho nefunkcionálních testů je odvozeno z funkcionálních. Oba typy používají stejné testovací případy, ale nefunkcionální kontrolují, zda je při dané funkcionalitě splněno také nějaké nefunkcionální omezení (např. že je vše vykonáno v určeném čase nebo zda může být daná funkčnost přenesena na novou platformu). Pozdní odhalení nefunkcionálního defektu může představovat vážnou hrozbu pro úspěch projektu.

Někdy je vhodné, aby nefunkcionální testování začalo už v rané fázi SDLC (např. jako součást revizí a při testování komponent nebo systémovém testování) a někdy je nutné využít velmi specifické testovací prostředí jako je například laboratoř pro testování použitelnosti.

Testování černé skříňky (viz kapitola 4.2) je založeno na specifikaci a odvozuje testy z dokumentace testovaného objektu (tedy ne z chování samotného objektu). Hlavním cílem testování černé skříňky je kontrola chování systému podle jeho specifikace.

Testování bílé skříňky (viz kapitola 4.3) je založeno na struktuře a odvozuje testy z implementace systému nebo z jeho vnitřní struktury, což je např. kód, architektura, pracovní toky (workflows) nebo datové toky. Hlavním cílem testování bílé skříňky je dosahovat dostatečného pokrytí testované struktury (např. kódu).

Všechny čtyři výše uvedené typy testů lze aplikovat ve všech úrovních testů, i když konkrétní implementace bude v každé úrovni odlišná. K odvození testovacích podmínek a testovacích případů pro všechny uvedené typy testů lze použít různé techniky testování.

2.2.3 Konfirmační a regresní testování

Pokud se v komponentě nebo systému provádějí změny, je obvykle jejich cílem rozšíření (přidáním nové funkcionality) nebo oprava (odstraněním defektu). Následné testování by poté mělo zahrnovat jak konfirmační testování, tak regresní testování.

Konfirmační testování ověřuje, zda byl původní defekt úspěšně opraven. V závislosti na riziku lze otestovat opravenou verzi softwaru několika různými způsoby, např.:

- Provedením všech testovacích případů, které dříve selhaly kvůli defektu.
- Přidáním nových testů s cílem pokrývat všechny změny potřebné k opravě defektu.

V případech, kdy je na opravu defektu málo času nebo finančních prostředků, může být konfirmační testování omezeno na pouhé provedení kroků, které by měly reprodukovat původní selhání způsobené defektem a zkontrolovat, zda k selhání nedochází.

Regresní testování ověřuje, zda nedošlo k žádným nežádoucím dopadům po realizovaných změnách, včetně oprav, které již byly potvrzeny konfirmačním testováním. Tyto nežádoucí dopady by mohly ovlivnit jak komponentu, kde byla změna provedena, tak i jiné komponenty ve stejném systému nebo dokonce jiné propojené systémy. Regresní testování nemusí být omezeno na samotný testovaný objekt, ale může také souviset s prostředím. Doporučuje se proto před testováním nejprve provést tzv. analýzu dopadu s cílem optimalizovat rozsah regresního testování. Analýza dopadu ukazuje, které části softwaru by mohly být ovlivněny.

Sady regresních testů se spouštějí mnohokrát a obvykle se jejich počet zvyšuje s každou novou iterací nebo vydáním, proto jsou vhodným kandidátem na automatizaci testování. Automatizace těchto testů by měla začít už v raných fázích projektu (viz kap. 6). Automatizované regresní testy je vhodné zařadit do CI pipeline (např. DevOps pipeline, viz kapitola 2.1.4). Regresní testy mohou být dle situace vykonávány v různých úrovních testování.

Konfirmační a/nebo regresní testování testovaného objektu je nutno použít v těch úrovních testování, kde došlo k opravám defektů a/nebo jsou v nich provedeny změny.

2.3 Testování údržby

Existují různé druhy údržby s různými cíli, např oprava, adaptace na změnu prostředí, zlepšení výkonu nebo zlepšení udržovatelnosti (viz norma ISO/IEC 14764). Údržba může být jak plánovaná, tak neplánovaná (hotfix). Před provedením změny lze taktéž provést analýzu dopadu, která pomůže při rozhodování, zda by změna měla být provedena, a to na základě potenciálních důsledků v jiných oblastech systému. Testování změn systému,

který je již v produkci, zahrnuje jak vyhodnocení úspěšnosti implementace změny, tak kontrolu možné regrese v nezměněných částech systému (což je obvykle většina systému).

Rozsah testování údržby závisí na těchto faktorech:

- míra rizika změny,
- velikost stávajícího systému,
- rozsah změny.

Spouštěče (aktivační události) údržby a testování údržby lze rozdělit takto:

- Úpravy jako jsou plánovaná vylepšení (např. v celém vydání), opravné změny nebo hotfixy.
- Aktualizace nebo migrace provozního prostředí (například z jedné platformy na jinou), kdy je nutné provést testy spojené s novým prostředím a změněným softwarem, případně testy konverze dat, kdy jsou data z jiné aplikace migrována do systému, který je udržován.
- Vyřazení aplikace z provozu na konci její životnosti. Při vyřazení aplikace nebo systému může být vhodné provést testování archivace dat, a to zejména pro taková data, pro která jsou požadována dlouhé lhůty archivace. Pro případ, kdy by bylo během doby archivace nutné načíst některá archivovaná data, je možné otestovat také procesy obnovení a načtení (po archivaci).

3 Statické testování – 80 minut

Klíčová slova

anomálie, dynamické testování, formální revize, inspekce, neformální revize, předvedení (walkthrough), revize, statická analýza, statické testování, technická revize

Studijní cíle

3.1 Základy statického testování

FL-3.1.1 (K1) Určit typy produktů, které mohou být otestovány s použitím různých technik statického testování.
FL-3.1.2 (K2) Vysvětlit hodnotu statického testování.
FL-3.1.3 (K2) Porovnat a uvést odlišnosti statického a dynamického testování.

3.2 Proces zpětné vazby a revize

FL-3.2.1 (K1) Identifikovat výhody včasné a časté zpětné vazby od zainteresovaných stran.
FL-3.2.2 (K2) Shrnout činnosti procesu revize.
FL-3.2.3 (K1) Vybavit si, které odpovědnosti při provádění revizí jsou přiřazeny jednotlivým rolím.
FL-3.2.4 (K2) Porovnat a uvést odlišnosti různých typů revizí.
FL-3.2.5 (K1) Vybavit si faktory, které přispívají k úspěšné revizi.

3.1 Základy statického testování

Na rozdíl od dynamického testování není při statickém testování nutné testovaný software spouštět. Kód, specifikace procesu, specifikace architektury systému nebo jiné pracovní produkty jsou hodnoceny a prověřovány buďto manuálně (např. revizemi) nebo pomocí nástroje (např. statická analýza). Mezi cíle testování patří zlepšování kvality, odhalování defektů a posuzování vlastností jako je čitelnost, úplnost, správnost, testovatelnost a konzistence. Statické testování lze používat jak pro ověřování (verifikaci), tak pro validaci.

Testeři, zástupci byznysu a vývojáři během popisu příkladů (např. při využití techniky specifikace pomocí příkladů), psaní uživatelských scénářů a zpřesňování backlogu (backlog refinement) spolupracují tak, aby uživatelské scénáře a související pracovní produkty splňovaly definovaná kritéria, např. definice připravenosti (viz kapitola 5.1.3). Techniky revizí lze použít k zajištění toho, aby uživatelské scénáře byly úplné a srozumitelné a obsahovaly testovatelná akceptační kritéria. Tím, že testeři kladou správné otázky vlastně zkoumají, konfrontují a pomáhají vylepšovat navrhované uživatelské scénáře.

Statická analýza může upozornit na problémy ještě před provedením dynamického testování, přičemž často vyžaduje menší pracnost – nevyžaduje totiž tvorbu testovacích případů a lze u ní využít různých nástrojů (viz kapitola 6). Statická analýza je často začleněna do nástrojů průběžně integrace (viz kapitola 2.1.4). I když se z velké části používá k odhalování specifických defektů kódu, lze ji také použít k vyhodnocení udržovatelnosti a bezpečnosti. Dalšími příklady nástrojů statické analýzy jsou nástroje pro kontrolu pravopisu a čitelnosti.

3.1.1 Pracovní produkty, které mohou být prověřeny statickým testováním

Téměř každý pracovní produkt může být prozkoumán pomocí statického testování. Mezi takové produkty patří například specifikace požadavků, zdrojový kód, plány testování, testovací případy, položky produktového backlogu, testovací listiny, projektová dokumentace, smlouvy nebo modely.

Revizi lze použít na jakýkoliv pracovní produkt, který lze přečíst a porozumět mu.

Pro **statickou analýzu** je nutné, aby měly zkoumané pracovní produkty formální strukturu, vůči které mohou být kontrolovány (např. modely, kód nebo text s formální syntaxí).

Pracovní produkty, které nejsou vhodné pro statické testování, jsou obvykle ty, které jsou obtížně interpretovatelné lidmi a současně by neměly být analyzovány nástroji (např. spustitelný kód třetí strany, který z právních důvodů nelze analyzovat s využitím nástrojů).

3.1.2 Přínosy statického testování

Statické testování může odhalit defekty v nejranějších fázích SDLC, čímž naplňuje princip včasného testování (viz kapitola 1.3). Může také odhalit defekty, které nelze zjistit dynamickým testováním (např. nedosažitelný kód, chybně implementované návrhové vzory, defekty v nespustitelných pracovních produktech).

Statické testování vytváří důvěru v dané pracovní produkty a umožňuje vyhodnotit jejich kvalitu. Ověřením zdokumentovaných požadavků mohou všichni zástupci zainteresovaných stran také zajistit, aby tyto požadavky odrážely jejich skutečné potřeby. Vzhledem k tomu, že statické testování může být prováděno v raných fázích SDLC, je možné nastavit pravidla jednotného chápání (např. požadavků), čímž se také zlepší komunikace mezi zainteresovanými stranami. Z tohoto důvodu se doporučuje zapojit do statického testování různé typy účastníků.

I když může být provedení revizí nákladné, celkové náklady na projekt jsou obvykle mnohem nižší, než když se revize vůbec neprovádí. Důvodem je to, že později v projektu není potřeba vynaložit tolik času a pracnosti na opravu defektů.

Některé defekty v kódu lze odhalit pomocí statické analýzy mnohem efektivněji než při dynamickém testování, což obvykle vede k jejich menšímu počtu a tím i nižší celkové pracnosti při vývoji.

3.1.3 Rozdíly mezi statickým a dynamickým testováním

Statické a dynamické testovací postupy se vzájemně doplňují. Mají podobné cíle (např. odhalování defektů v pracovních produktech, viz kapitola 1.1.1), ale existují mezi nimi také určité rozdíly, například:

- Jak statické, tak dynamické testování (včetně analýzy selhání) může vést k odhalení defektu, nicméně existují některé typy defektů, které lze nalézt buď jenom statickým nebo jenom dynamickým testováním.
- Statické testování najde defekty přímo, zatímco dynamické testování hledá selhání, ze kterých jsou příslušné defekty určeny následnou analýzou.
- Statické testování může snadněji odhalovat defekty obtížně dosažitelné pomocí dynamického testování nebo takové, které jsou v kódu v částech spouštěných jen zřídka.
- Dynamické testování lze použít pouze na spustitelné pracovní produkty, statické testování i na nespustitelné.
- Statické testování lze použít k měření kvalitativních charakteristik (např. udržovatelnosti) nezávislých na spuštění kódu, dynamické testování pouze k těm, které jsou na spuštění kódu závislé (např. výkonnostní efektivity).

Mezi typické defekty odhalitelné snadněji a levněji statickým testováním patří:

- defekty v požadavcích (např. nesrovnalosti, nejednoznačnosti, rozpory, opomenutí, nepřesnosti a duplicity),
- defekty v návrhu (např. neefektivní databázové struktury, špatná modularita),
- některé typy defektů při programování (např. proměnné s nedefinovanými hodnotami, nedeklarované proměnné, nedosažitelný nebo duplicitní kód, nadměrná složitost kódu),
- odchylky od norem (např. nedostatečné dodržování konvencí pro programování),
- nesprávné specifikace rozhraní (např. rozdílný počet, typ nebo pořadí parametrů),
- některé bezpečnostní zranitelnosti (např. přetečení vyrovnávací paměti),
- chybějící části nebo nepřesnosti v pokrytí testovací báze (např. chybějící testy pro některé akceptační kritérium).

3.2 Proces zpětné vazby a revize

3.2.1 Výhody včasné a časté zpětné vazby od zainteresovaných stran

Včasná a častá zpětná vazba pomáhá s odhalením potenciálních problémů s kvalitou. Pokud se zainteresované strany zapojují do vývoje nedostatečně, vyvíjený produkt nemusí splňovat jejich původní nebo současné představy. Pokud není tým schopen dodat to, co zainteresované strany chtějí, hrozí riziko vícenákladů na přepracování, zmeškaných termínů, vzájemného obviňování, a dokonce může dojít k úplnému selhání projektu.

Častá zpětná vazba od zainteresovaných stran během SDLC může zabránit nedorozuměním při definici požadavků a zajistit, aby změny požadavků byly správně a včas pochopeny a implementovány. To pomáhá vývojovému týmu lépe porozumět tomu, co vyvíjí. Umožňuje jim také zaměřit se na ty užitné vlastnosti, které přinášejí zainteresovaným stranám nejvyšší přidanou hodnotu, a které mají nejvíce pozitivní dopad na zjištěná rizika.

3.2.2 Činnosti procesu revize

Norma ISO/IEC 20246 definuje obecný proces revize a popisuje strukturovaný, ale flexibilní rámec, ze kterého může být daný proces revize přizpůsoben konkrétní situaci. Čím vyšší je požadovaný stupeň formálnosti revize, tím vyšší je počet úkolů během různých činností.

Mnohé pracovní produkty jsou tak rozsáhlé, že je nelze pokryt pouze jedinou revizí a proces revize může být proveden opakovaně.

Činnosti procesu revize jsou:

- **Plánování**. Během fáze plánování se stanoví rozsah revize obsahující definici účelu a revidovaného pracovního produktu, kvalitativní charakteristiky pro vyhodnocení, oblasti zájmu, výstupní kritéria, doplňující informace (např. normy), pracnost a časový rámec celé revize.
- **Zahájení revize**. Během zahájení revize je cílem zajistit, aby bylo všechno (včetně všech zainteresovaných stran) připraveno k revizi. To mimo jiné vyžaduje, aby měl každý účastník přístup k revidovanému pracovnímu produktu, rozuměl své roli a odpovědnostem a obdržel vše potřebné k provedení revize.
- **Individuální revize**. Každý revidující reviduje definovaný pracovní produkt individuálně s cílem posoudit jeho kvalitu a identifikovat anomálie, doporučení a otázky pomocí jedné nebo více technik revize (např. revize založená na kontrolním seznamu, revize založená na scénáři – viz norma ISO/IEC 20246). Revidující zaznamenává všechny zjištěné anomálie, doporučení a otázky.
- **Komunikace a analýza**. Ne každá anomálie identifikovaná během revize musí být nutně defekt. Proto je nutné všechny takové anomálie analyzovat a prodiskutovat a u každé by mělo být rozhodnuto o jejím stavu, vlastnictví a požadovaných opatřeních. To se obvykle provádí při revizní schůzce, během níž účastníci také diskutují o úrovni kvality každého revidovaného pracovního produktu a jaká následná opatření jsou vyžadována. K uzavření opatření může být vyžadována další revize.
- **Opravy a reportování**. Aby bylo možné provést nápravu, měl by být pro každý defekt vytvořen report o defektu. Při splnění daných výstupních kritérií může být pracovní produkt akceptován a výsledky revize jsou reportovány.

3.2.3 Role a odpovědnosti při revizích

Revizi mohou provádět různí lidé a mohou zastávat různé role. Hlavní role a jejich povinnosti jsou:

- **Manažer** – rozhoduje o tom, co má být revidováno a jaké zdroje budou využity (včetně lidí a času).
- **Autor** – vytváří a opravuje revidovaný pracovní produkt.
- **Moderátor** (někdy také facilitátor) – zajišťuje efektivní průběh revizních schůzek včetně využití mediace. Mimo jiné dohlíží na dodržení časového rámce a zajištění komfortního prostředí, ve kterém může každý svobodně vyjádřit svůj názor.
- **Zapisovatel** – shromažďuje anomálie od revidujících a zaznamenává další informace z revize jako např. různá rozhodnutí nebo nové anomálie zjištěné během revizní schůzky.
- **Revidující** – provádí revizi. Revidujícím může být někdo, kdo pracuje na projektu nebo je odborníkem na danou problematiku nebo i kterákoliv jiná zainteresovaná strana.
- **Vedoucí revize** – přebírá celkovou odpovědnost za revizi (např. rozhodování o tom, kdo do ní bude zapojen) a organizaci toho, kdy a kde se bude revize konat.

Popis dalších možných rolí lze nalézt v normě ISO/IEC 20246.

3.2.4 Typy revizí

Existuje mnoho typů revizí, od neformálních až po velmi formální. Požadovaná úroveň formálnosti závisí na faktorech jako je použitý SDLC, vyspělost procesu vývoje, kritičnost a složitost revidovaného pracovního produktu, právní nebo regulatorní požadavky a potřeba doložení záznamů pro případný audit. Stejný pracovní produkt může být revidován různými typy revizí, např. nejprve neformální a později formálnější.

Výběr správného typu revize je klíčový pro dosažení požadovaných cílů revize (viz kapitola 3.2.5). Výběr je ale založen na dalších faktorech jako jsou potřeby projektu, dostupné zdroje, typ pracovního produktu, typ rizika, byznysová doména a firemní kultura.

Mezi běžně používané typy revizí patří:

- **Neformální revize**. Neformální revize se neřídí definovaným procesem a nevyžadují formální dokumentovaný výstup. Hlavním cílem je odhalování anomálií.
- **Předvedení (walkthrough)**. Předvedení (vedené autorem) může sloužit mnoha cílům jako je hodnocení kvality a budování důvěry v pracovní produkt, vzdělávání revidujících, dosažení dohody, generování nových nápadů, motivace a podpora autora s cílem zlepšovat pracovní produkt a odhalovat anomálie. Revidující mohou (ale nemusí) před předvedením provést individuální revizi.

- **Technická revize**. Technickou revizi provádějí odborně kvalifikovaní revidující a vede ji moderátor. Cílem technické revize je primárně dosáhnout shody a učinit rozhodnutí týkající se nějakého technického problému, ale také odhalit anomálie, vyhodnotit kvalitu, vybudovat důvěru v pracovní produkt, generovat nové nápady, motivovat autory a podpořit je ve zlepšování.
- **Inspekce**. Vzhledem k tomu, že inspekce jsou nejformálnějším typem revize, řídí se komplexním obecným procesem (viz kapitola 3.2.2). Hlavním cílem je nacházet maximální počet anomálií. Dalšími cíli jsou hodnocení kvality, budování důvěry v pracovní produkt, motivace a podpora autorů ve zlepšování. Jsou shromažďovány metriky, které se používají ke zlepšování celého SDLC včetně samotného procesu inspekce. Při inspekcích nemůže autor vystupovat jako vedoucí revize nebo zapisovatel.

3.2.5 Faktory úspěchu při revizi

Existuje několik faktorů klíčových pro úspěch procesu revize:

- Jsou definovány jasné cíle a měřitelná výstupní kritéria. Nejsou hodnoceni účastníci revize, ale revidovaný pracovní produkt.
- Je vybrán takový typ revize, který je vhodný pro dosažení daných cílů, pro typ pracovního produktu, pro účastníky revize a pro potřeby a kontext projektu.
- Revize jsou prováděné po malých částech, takže revidující neztrácejí koncentraci během revize a/nebo během revizních schůzek (pokud se konají).
- Zainteresovaným stranám a autorům je poskytována zpětná vazba z revizí tak, aby mohli zlepšovat produkt a své činnosti (viz kapitola 3.2.1).
- Účastníci mají dostatek času na přípravu.
- Management podporuje proces revize.
- Revize jsou součástí firemní kultury s cílem podporovat učení a zlepšování procesů.
- Je poskytnuto vhodné odborné školení pro všechny účastníky tak, aby věděli, jak plnit svou roli v procesu.
- Schůzky jsou správně řízené.

4 Testovací analýza a návrh testů – 390 minut

Klíčová slova

akceptační kritéria, analýza hraničních hodnot, odhadování chyb, pokrytí, pokrytí příkazů, pokrytí větví, položka pokrytí, průzkumné testování, technika testování, technika testování bílé skříňky, technika testování černé skříňky, technika testování založená na zkušenostech, testovací přístup založený na spolupráci, testování dle rozhodovací tabulky, testování přechodů stavů, testování založené na kontrolních seznamech, rozdělení tříd ekvivalence, vývoj řízený akceptačními testy

Studijní cíle

4.1 Přehled technik testování

FL-4.1.1 (K2) Rozlišit mezi technikami testování černé skříňky, bílé skříňky a technikami založenými na zkušenostech.

4.2 Techniky testování černé skříňky

FL-4.2.1 (K3) Použít techniku rozdělení tříd ekvivalence k odvození testovacích případů.
FL-4.2.2 (K3) Použít techniku analýzy hraničních hodnot k odvození testovacích případů.
FL-4.2.3 (K3) Použít techniku testování dle rozhodovací tabulky k odvození testovacích případů.
FL-4.2.4 (K3) Použít techniku testování přechodů stavů k odvození testovacích případů.

4.3 Techniky testování bílé skříňky

FL-4.3.1 (K2) Vysvětlit techniku testování příkazů.
FL-4.3.2 (K2) Vysvětlit techniku testování větví.
FL-4.3.3 (K2) Vysvětlit přínos testování bílé skříňky.

4.4 Techniky testování založené na zkušenostech

FL-4.4.1 (K2) Vysvětlit techniku odhadování chyb.
FL-4.4.2 (K2) Vysvětlit techniku průzkumného testování.
FL-4.4.3 (K2) Vysvětlit techniku testování založenou na kontrolních seznamech.

4.5. Přístupy k testování založené na spolupráci

FL-4.5.1 (K2) Vysvětlit, jak psát uživatelské scénáře ve spolupráci s vývojáři a zástupci byznysu.
FL-4.5.2 (K2) Kategorizovat různé možností psaní akceptačních kritérií.
FL-4.5.3 (K3) Použít techniku vývoje řízeného akceptačními testy k odvození testovacích případů.

4.1 Přehled technik testování

Techniky testování podporují testery při testovací analýze (ve smyslu „co testovat“) a při návrhu testů (ve smyslu „jak testovat“) a pomáhají systematicky definovat relativně malou, ale dostatečnou sadu testovacích případů. Pomáhají také testerům (opět během testovací analýzy a návrhu testů) definovat testovací podmínky, identifikovat položky pokrytí a identifikovat testovací data. Více informací o technikách testování lze nalézt v normě ISO/IEC/IEEE 29119-4 a v (Beizer 1990, Craig 2002, Copeland 2004, Koomen 2006, Jorgensen 2014, Ammann 2016, Forgács 2019).

V těchto učebních osnovách jsou popsány techniky testování černé skříňky, testování bílé skříňky a testování založené na zkušenostech.

Techniky testování černé skříňky (známé také jako techniky založené na specifikacích) jsou založeny na analýze specifikovaného chování testovaného objektu bez informací o jeho vnitřní struktuře. Testovací případy jsou tedy nezávislé na tom, jak je software napsán. V důsledku toho platí, že pokud se implementace kódu změní, ale požadované chování zůstane stejné, lze testovací případy stále používat.

Techniky testování bílé skříňky (známé také jako techniky založené na struktuře) jsou založeny na analýze vnitřní struktury a způsobu zpracování uvnitř testovaného objektu. Vzhledem k tomu, že testovací případy jsou závislé na návrhu daného softwaru, mohou být vytvořeny pouze po ukončení návrhu nebo implementaci testovaného objektu (pokud není použita technika, která to umožňuje, např. TDD).

Techniky testování založené na zkušenostech využívají znalosti a zkušenosti testerů pro návrh a implementaci testovacích případů. Efektivita těchto technik silně závisí na dovednostech testerů. Techniky testování založené na zkušenostech mohou odhalit defekty, které techniky testování černé a bílé skříňky neodhalí. Lze tedy říct, že techniky testování založené na zkušenostech jsou doplňkem technik testování černé a bílé skříňky.

4.2 Techniky testování černé skříňky

Běžně používané techniky testování černé skříňky jsou popsány v následujících kapitolách:

- rozdělení tříd ekvivalence,
- analýza hraničních hodnot,
- testování dle rozhodovací tabulky,
- testování přechodů stavů.

4.2.1 Rozdělení tříd ekvivalence.

Rozdělení tříd ekvivalence (EP – equivalence partitioning) je technika testování rozdělující data do tříd (označovaných jako třídy ekvivalence), u kterých lze očekávat stejný způsob zpracování testovaným objektem. Teorie stojící za touto technikou spočívá v tom, že pokud testovací případ detekuje defekt pro některou hodnotu z určité třídy ekvivalence, měl by tento testovací případ odhalit defekt i pro jakoukoli jinou hodnotu téže třídy.

Třídu ekvivalence lze identifikovat pro libovolný datový element související s testovaným objektem jako jsou např. vstupy, výstupy, konfigurační položky, interní hodnoty, hodnoty související s časem nebo parametry rozhraní. Třídy mohou být spojité nebo diskrétní, uspořádané nebo neuspořádané, konečné nebo nekonečné. Třídy se nesmí překrývat a musí se jednat o neprázdné množiny.

Pro jednoduché testované objekty může být aplikace této techniky snadná, ale v praxi je pochopení toho, jak bude testovaný objekt reagovat s různými hodnotami, často komplikované. Proto by se rozdělení do tříd mělo provádět uvážlivě.

Třída ekvivalence, která obsahuje platné hodnoty, se nazývá platná třída ekvivalence. Třída ekvivalence, která obsahuje neplatné hodnoty, se nazývá neplatná třída ekvivalence. Definice platných a neplatných hodnot se mohou v praxi výrazně lišit v závislosti na tom, jaké týmy a jaké organizace s nimi pracují. Například platné hodnoty mohou být interpretovány jako ty, které by měly být zpracovány testovaným objektem, nebo jako ty, pro které specifikace definuje jejich zpracování. Neplatné hodnoty mohou být interpretovány jako ty, které by

měly být ignorovány nebo odmítnuty testovaným objektem, nebo jako ty, pro které není ve specifikaci testovaného objektu definováno žádné zpracování.

V EP jsou položkami pokrytí samotné třídy ekvivalence. Pro dosažení 100% pokrytí musí testovací případy pokrývat všechny identifikované třídy ekvivalence (včetně neplatných), a to použitím minimálně jedné hodnoty z každé třídy. Pokrytí je měřeno jako poměr počtu tříd otestovaných alespoň jedním testovacím případem k celkovému počtu identifikovaných tříd (obvykle vyjádřeno v procentech).

Mnoho testovaných objektů obsahuje více množin tříd ekvivalence (např. testované objekty s více vstupními parametry). V tomto případě může jeden testovací případ pokrývat třídy z různých množin tříd. Nejjednodušším kritériem pokrytí v takových případech je tzv. pokrytí každé volby (viz Ammann 2016). Toto kritérium vyžaduje, aby testovací případy pokryly každou třídu z každé množiny tříd alespoň jednou a zároveň nebere v úvahu kombinace tříd.

4.2.2 Analýza hraničních hodnot

Analýza hraničních hodnot (BVA – boundary value analysis) je technika založená na pokrytí okrajových hodnot tříd ekvivalencí. Proto ji lze použít pouze pro uspořádané (seřazené) třídy, kdy minimální a maximální hodnoty každé třídy jsou její hraniční hodnoty. Pro tuto techniku platí, že pokud dva prvky patří do stejné třídy, musí do této třídy patřit také všechny prvky ležící mezi nimi.

BVA se zaměřuje na hraniční hodnoty tříd, protože pravděpodobnost, že vývojáři udělají chybu právě na těchto hranicích, je vyšší. Typické defekty zjištěné technikou BVA se nacházejí v těch oblastech, kde jsou implementované (skutečné) hranice přesunuty na pozice nad nebo pod jejich specifikovanými (předpokládanými) hodnotami, případně kde jsou zcela vynechány.

Tyto učební osnovy se zabývají dvěma variantami: 2-hodnotová a 3-hodnotová BVA. Tyto dvě varianty se liší v počtu položek pokrytí v dané hranici nutných k dosažení 100% pokrytí.

V případě 2-hodnotové BVA (Craig 2002, Myers 2011) existují pro každou hranici dvě položky pokrytí: hraniční hodnota a její nejbližší soused patřící do sousední třídy. Pro dosažení 100% pokrytí s touto variantou musí být testovací případy vykonány pro všechny položky pokrytí (tj. pro všechny identifikované hraniční hodnoty). Pokrytí se měří jako poměr počtu hraničních hodnot pokrytých testovacími případy k celkovému počtu identifikovaných hraničních hodnot (vyjádřeno v procentech).

V případě 3-hodnotové BVA (Koomen 2006, O'Regan 2019) existují pro každou hranici tři položky pokrytí: hraniční hodnota a oba její sousedé. U této varianty se může stát, že některé položky pokrytí nemusí být hraničními hodnotami žádné třídy. Pro dosažení 100% pokrytí s touto variantou musí být testovací případy vykonány pro všechny položky pokrytí, tj. pro identifikované hraniční hodnoty i pro všechny jejich sousedy. Pokrytí se měří jako poměr součtu hraničních a jejích sousedních hodnot pokrytých testovacími případy k celkovému součtu identifikovaných hraničních hodnot a jejích sousedů (obvykle vyjádřeno v procentech).

3-hodnotová BVA je „přísnější“ než 2-hodnotová BVA, s její pomocí lze odhalit defekty, které by mohly být s použitím 2-hodnotové BVA přehlédnuty. Pokud je např. podmínka "IF (x ≤ 10) ..." nesprávně implementována jako "IF (x = 10) ...", žádný z testovacích případů vytvořených pomocí 2-hodnotové BVA (x = 10, x = 11) nemůže tuto chybnou implementaci (a následný defekt) odhalit. Testovací případ x = 9 odvozený pomocí 3-hodnotové BVA tuto chybu pravděpodobně odhalí.

4.2.3 Testování dle rozhodovací tabulky

Rozhodovací tabulky jsou vhodné pro testování implementace systémových požadavků definujících, jakým způsobem vedou různé kombinace podmínek k různým výsledkům. Představují efektivní způsob zaznamenávání složité logiky jako jsou například byznysová pravidla.

Při vytváření rozhodovacích tabulek se určí podmínky a výsledné akce systému, které tvoří dvě skupiny řádků tabulky. Každý sloupec odpovídá jednomu pravidlu rozhodování, které definuje jedinečnou kombinaci podmínek vedoucí k provedení akcí spojených s tímto pravidlem. V rozhodovacích tabulkách s omezeným počtem vstupů (limited-entry decision table) jsou všechny hodnoty podmínek a akcí (s výjimkou irelevantních nebo neproveditelných hodnot, viz níže) zobrazeny jako logické hodnoty (pravda / true nebo nepravda / false).

V rozhodovacích tabulkách s rozšířeným počtem vstupů (extended-entry decision table) mohou některé (nebo všechny) podmínky a akce také zpracovávat více hodnot (např. rozsahy hodnot, třídy ekvivalence, diskrétní hodnoty).

Symboly používané v rozhodovacích tabulkách pro podmínky jsou následující:

- „T“ (true) – podmínka je splněna,
- „F“ (false) – podmínka není splněna,
- „–“ znamená, že hodnota podmínky je pro výsledek akce (výstup) irelevantní,
- „N/A“ – podmínka je pro dané pravidlo neproveditelná.

Zápis akcí (výstupů) je následující:

- „X“ – akce by měla nastat,
- „“ (prázdná hodnota) – akce by neměla nastat.

Lze samozřejmě použít i jiné notace.

Úplná rozhodovací tabulka obsahuje takový počet sloupců, aby došlo k pokrytí každé kombinace podmínek. Tabulku lze zjednodušit odstraněním sloupců obsahujících neproveditelnou kombinaci podmínek nebo také sloučením sloupců, jejichž hodnoty nemají vliv na výsledek. Popis těchto algoritmů je ale nad rámec těchto osnov.

U rozhodovacích tabulek jsou položkami pokrytí sloupce obsahující proveditelnou kombinaci podmínek. Pro dosažení 100% pokrytí touto technikou musí testovací případy pokrýt všechny takové sloupce. Pokrytí se měří jako poměr počtu pokrytých sloupců k celkovému počtu proveditelných sloupců (obvykle vyjádřeno v procentech).

Význam testování dle rozhodovací tabulky spočívá v tom, že poskytuje systematický přístup k identifikaci všech kombinací podmínek, z nichž některé by jinak mohly být přehlédnuty. Pomáhá také při odhalování případných nedostatků v požadavcích.

V případě, že existuje mnoho podmínek, může být prověřování všech rozhodovacích pravidel časově náročné, protože počet pravidel roste exponenciálně s počtem podmínek. V takovém případě lze ke snížení počtu testovaných pravidel použít zjednodušenou rozhodovací tabulku nebo přístup založený na rizicích.

4.2.4 Testování přechodů stavů

Diagram přechodů stavů modeluje chování systému zobrazením jeho možných stavů a platných přechodů mezi nimi. Přechod je iniciován výskytem události a může být doplněn o podmínku přechodu (guard condition), jejíž splnění přechod podmiňuje. Předpokládá se, že přechody jsou prováděny okamžitě a někdy mohou vést k tomu, že software provede určitou akci. Běžná syntaxe označování přechodů je „událost [podmínka přechodu] / akce“. Pokud podmínky přechodů nebo akce neexistují (příp. jsou pro testery irelevantní), mohou být vynechány.

K diagramu přechodů stavů je ekvivalentní také tabulka přechodů stavů. Její řádky představují stavy a její sloupce události doplněné o podmínky přechodů (pokud existují). Jednotlivé položky tabulky (buňky) představují přechody a obsahují cílový stav spolu s podmínkami přechodů a výslednými akcemi (pokud jsou definovány). Na rozdíl od diagramu přechodů stavů zobrazuje tabulka přechodů stavů i neplatné přechody, které jsou reprezentovány prázdnými buňkami.

Testovací případ odvozený z diagramu přechodů stavů nebo z tabulky přechodů stavů je obvykle znázorněn jako série událostí vedoucí k posloupnosti změn stavů (a akcí, pokud jsou definovány). Jeden testovací případ může (a obvykle bude) pokrývat několik přechodů mezi stavy.

Existuje mnoho kritérií pokrytí pro testování přechodů stavů. Tyto učební osnovy popisují tři z nich.

Při **pokrytí všech stavů** jsou položkami pokrytí příslušné stavy. Aby bylo dosaženo 100% pokrytí všech stavů, musí testovací případy zajistit pokrytí všech stavů. Pokrytí se měří jako poměr počtu pokrytých stavů k celkovému počtu stavů (obvykle vyjádřeno v procentech).

Při **pokrytí platných přechodů** (nazývaném také pokrytí 0-switch) jsou položkami pokrytí jednotlivé platné přechody. Pro dosažení 100% pokrytí platných přechodů musí testovací případy pokrýt všechny platné přechody.

Pokrytí se měří jako poměr počtu pokrytých platných přechodů k celkovému počtu platných přechodů (obvykle vyjádřeno v procentech).

Při **pokrytí všech přechodů** jsou položkami pokrytí všechny přechody v tabulce přechodů stavů. Pro dosažení 100% pokrytí všech přechodů musí testovací případy prověřit všechny platné přechody a pokusit se prověřit přechody neplatné (tzn. pokrýt jak platné, tak neplatné přechody). Je vhodné testovat jeden neplatný přechod v jednom testovacím případu, neboť se tak snižuje riziko tzv. maskování defektu, tj. situace, kdy není možno identifikovat defekt v důsledku výskytu jiného. Pokrytí se měří jako počet platných a neplatných přechodů, které byly nebo měly být pokryty provedenými testovacími případy, k celkovému počtu všech platných a neplatných přechodů (obvykle vyjádřeno v procentech).

Kritérium pokrytí všech stavů je „slabší“ než kritérium pokrytí platných přechodů, protože může být obvykle dosaženo bez provedení všech přechodů. Nejpoužívanějším kritériem pokrytí je ale pokrytí platných přechodů, které by také mělo být minimálním požadavkem pro bezpečnostně kritický software.

Platí, že dosažení plného pokrytí platných přechodů garantuje také plné pokrytí všech stavů, dosažení plného pokrytí všech přechodů zaručuje jak plné pokrytí všech stavů, tak plné pokrytí platných přechodů.

4.3 Techniky testování bílé skříňky

Tato kapitola se zaměřuje na dvě nejznámější techniky testování bílé skříňky, s jejichž pomocí lze otestovat kód:

- testování příkazů,
- testování větví.

Existují samozřejmě pokročilejší techniky používané např. při testování v bezpečnostně kritických, provozně kritických a integračně kritických prostředích, kdy je cílem dosahovat vyššího stupně pokrytí. Techniky testování bílé skříňky nesouvisí výhradně s úrovní testování komponent. Existují také techniky testování bílé skříňky používané ve vyšších úrovních testování (např. při API testování) nebo dokonce techniky pro měření pokrytí nesouvisející s kódem (např. pokrytí neuronů při testování neuronových sítí). Popis všech takových technik přesahuje rámec daný těmito učebním osnovami.

4.3.1 Testování a pokrytí příkazů

Při testování příkazů jsou položkami pokrytí spustitelné příkazy. Cílem je navrhovat takové testovací případy, které pokryjí příkazy v určité části kódu při definované akceptovatelné úrovni pokrytí. Pokrytí se měří jako počet spustitelných příkazů pokrytých testy k celkovému počtu spustitelných příkazů v kódu (obvykle vyjádřeno v procentech).

V případě dosažení 100% pokrytí příkazů je zajištěno, že všechny spustitelné příkazy v kódu byly otestovány alespoň jednou. To znamená, že bude spuštěn každý příkaz, který může způsobit selhání indikující výskyt defektu. Opět ale platí, že otestování příkazu pomocí testovacího případu nemusí vždy odhalit defekt, technika například nemusí detekovat defekty související s daty (např. dělení nulou, které selže pouze při nulové hodnotě dělitele). 100% pokrytí příkazů také nezaručuje, že veškerá rozhodovací logika byla otestována, protože nemusí dojít ke spuštění všech větví v kódu (viz kapitola 4.3.2).

4.3.2 Testování a pokrytí větví

Větev je přenos řízení mezi dvěma uzly v grafu řídicího toku zobrazujícího možné sekvence, ve kterých jsou příkazy zdrojového kódu v testovacím objektu prováděny. Každý přenos řízení může být buď nepodmíněný (tj. lineární kód) nebo podmíněný (tj. výsledek rozhodnutí).

Při testování větví jsou položkami pokrytí větve a cílem je navrhovat testovací případy tak, aby došlo k pokrytí větví v určité části kódu při definované akceptovatelné úrovni pokrytí. Pokrytí se měří jako počet větví pokrytých testy k celkovému počtu větví (obvykle vyjádřeno v procentech).

Při dosažení 100% pokrytí větví jsou všechny nepodmíněné a podmíněné větve v kódu pokryty testovacími případy. Podmíněné větve obvykle odpovídají pravdivému nebo nepravdivému výsledku z podmínky (IF / THEN / ELSE), výsledku z vícecestné podmínky (SWITCH / CASE) nebo rozhodnutí, zda ukončit nebo pokračovat ve

smyčce (FOR / NEXT, WHILE / DO, DO / WHILE, REPEAT / UNTIL). Opět ale platí, že otestování větve pomocí testovacího případu nemusí vždy odhalit defekt, například nemusí detekovat defekt vyžadující provedení určité cesty v kódu.

Pokrytí větví automaticky zaručuje pokrytí příkazů. To znamená, že jakákoliv sada testovacích případů, která dosáhne 100% pokrytí větví, dosáhne také 100% pokrytí příkazů (ale ne naopak).

4.3.3 Význam testování bílé skříňky

Silnou stránkou technik bílé skříňky je, že při testování je zohledněna samotná implementace softwaru, což usnadňuje detekci defektů v případech, kdy je jeho specifikace vágní, zastaralá nebo neúplná. Slabinou je fakt, že nemusí odhalit defekty při opomenutí implementace jednoho nebo více požadavků (Watson 1996).

Techniky bílé skříňky mohou být použity při statickém testování (např. během tzv. běhů kódu „nanečisto“, dry-runs). Jsou také vhodné při revizi kódu, který ještě není připraven ke spuštění (Hetzel 1988) stejně jako při revizi pseudokódu nebo jiného zápisu logiky vyšší nebo nižší úrovně, které mohou být modelovány pomocí grafu řídicího toku.

Pouhé testování pomocí technik černé skříňky neposkytuje informace o míře skutečného pokrytí kódu. Naopak výsledky testování s využitím technik bílé skříňky poskytují objektivní měření pokrytí a poskytují také nezbytné informace umožňující vytváření dalších testů s cílem zvyšování tohoto pokrytí a tím i následné zvyšování důvěry v kód.

4.4 Techniky testování založené na zkušenostech

Nejznámější používané techniky testování založené na zkušenostech popsané v následujících kapitolách jsou:

- odhadování chyb,
- průzkumné testování,
- testování založené na kontrolním seznamu.

4.4.1 Odhadování chyb

Odhadování chyb je technika používaná k předvídání výskytu chyb, defektů a selhání, která je založená na znalostech testerů jako jsou např.:

- jak fungovala aplikace v minulosti,
- typické vývojářské chyby a typické defekty, které jsou důsledkem těchto chyb,
- typická selhání, která nastala v jiných (podobných) aplikacích.

Obecně platí, že chyby, defekty a selhání mohou souviset se vstupem (např. nebyl akceptován správný vstup, parametry byly chybně nastaveny nebo úplně chyběly), výstupem (např. chybný formát nebo výsledek), logikou (např. chybějící ošetření některých případů v příkazu switch/case, chybný operátor), výpočtem (např. nesprávný operand, chybný výpočet), rozhraními (např. nesoulad parametrů, nekompatibilní typy) nebo daty (např. nesprávná inicializace, chybný typ).

Známým metodickým přístupem k implementaci odhadování chyb je útok na vady (fault attack). Při této technice je nutné, aby testeři vytvořili nebo získali seznam možných chyb, defektů a selhání a navrhli takové testy, které je identifikují, odhalí nebo způsobí. Tyto seznamy lze sestavit na základě zkušeností, údajů o defektech a selháních nebo z obecných znalostí o důvodech selhávání softwaru.

Více informací o těchto technikách viz Whittaker 2002, Whittaker 2003 a Andrews 2006.

4.4.2 Průzkumné testování

Při průzkumném testování jsou testy současně navrhovány, prováděny a vyhodnocovány, zatímco testeři získávají znalosti o testovaném objektu. Testování se (mimo jiné) používá k získání dalších informací o testovaném objektu, k jeho hlubšímu prozkoumání pomocí cílených testů a k vytvoření testů pro dosud neotestované oblasti.

Průzkumné testování se někdy provádí pomocí tzv. testování v relacích s cílem lépe strukturovat testování, kdy se definuje časový rámec pro test. Jako vstup pro řízení celého testu používají testeři tzv. testovací listinu (test charter) s definicí cílů testování. Po testovací relaci obvykle následuje tzv. debriefing s diskusí testerů a zástupců zainteresovaných stran, kteří se zajímají o výsledky testovací relace.

Obecnými testovacími podmínkami mohou být při průzkumném testování cíle testování. Jednotlivé položky pokrytí jsou identifikovány a prověřovány během testovacích relací a testeři mohou používat dokumenty testovací relace (test session sheets) pro záznam prováděných kroků a objevených zjištění.

Průzkumné testování je nejvíce užitečné v případech, kdy je specifikace nedostatečná nebo úplně chybí, taktéž je vhodnou metodou v případě výrazného časové tlaku na testování. Je také užitečné jako doplněk jiných (formálnějších) testovacích technik. Obecně bude účinnější v případech, kdy jsou testeři zkušení, mají znalosti z dané domény a vysokou úroveň základních dovedností jako jsou analytické schopnosti, zvídavost a kreativita (viz kapitola 1.5.1).

Průzkumné testování může využívat i jiné testovací techniky (např. rozdělení tříd ekvivalence). Více informací o průzkumném testování viz Kaner 1999, Whittaker 2009 a Hendrickson 2013.

4.4.3 Testování založené na kontrolních seznamech

Při testování založeném na kontrolních seznamech testeři navrhují, implementují a provádí testy za účelem pokrytí testovacích podmínek uvedených v kontrolním seznamu. Kontrolní seznamy lze vytvářet na základě zkušeností, znalostí o tom, co je pro uživatele důležité, nebo na základě pochopení, proč a jak software selhává. Kontrolní seznamy by neměly obsahovat položky, které lze zkontrolovat automaticky, položky, které jsou vhodnější jako vstupní či výstupní kritéria, nebo položky, které jsou příliš obecné (Brykczynski 1999).

Položky kontrolního seznamu jsou často formulovány ve formě otázky. Každou položku by mělo být možné přímo ověřit, a to odděleně (nezávisle na ostatních položkách). Položky kontrolního seznamu se mohou týkat požadavků, vlastností grafického rozhraní (UI), kvalitativních charakteristik nebo jiných forem testovacích podmínek. Kontrolní seznamy mohou být taktéž vytvořeny jako podpora různých typů testů včetně funkcionálního a nefunkcionálního testování (např. 10 principů testování použitelnosti, viz Nielsen 1994).

Některé položky kontrolního seznamu mohou být časem méně efektivní, protože vývojáři se postupně učí a neopakují určitý typ chyb. Kontrolní seznamy by proto měly být pravidelně aktualizovány na základě analýzy defektů, typicky je možné do nich přidávat nové položky reflektující výskyt nově zjištěných defektů s vysokou závažností. Je však třeba dbát na to, aby se kontrolní seznam nestal příliš dlouhým (Gawande 2009).

V případech absence testovacích případů může testování založené na kontrolních seznamech poskytnout návod pro testování a zajistit určitý stupeň konzistence. Jelikož se jedná o obecné seznamy (high-level sheets), je pravděpodobné, že se v takovém testování může objevit určitá variabilita. To může přispět k většímu pokrytí, ale také k menší míře opakovatelnosti testování.

4.5 Přístupy k testování založené na spolupráci

Každá z výše uvedených technik (viz kapitoly 4.2, 4.3, 4.4) má svůj specifický cíl ve vztahu k identifikaci defektů. Přístupy založené na spolupráci využívají odlišné principy a jsou zaměřeny spíše na prevenci výskytu defektů využitím spolupráce a komunikace.

4.5.1 Společné psaní uživatelských scénářů

Uživatelský scénář (user story) představuje užitnou vlastnost (feature), která má pro uživatele nebo pro budoucího majitele softwaru / systému určitou hodnotu. Uživatelské scénáře obsahují tři důležité části (viz Jeffries 2000) nazývané dohromady „3C“:

- **karta** (card) – médium popisující uživatelský scénář (např. barevné štítky na tabuli nebo digitální karta v online systému),
- **konverzace** (conversation) – vysvětlení, jak bude software používán (může být dokumentována nebo být pouze verbální),

- **potvrzení** (confirmation) – ve formě akceptačního kritéria (viz kapitola 4.5.2).

Nejběžnějším formátem uživatelského scénáře je formule „Jako [ROLE] chci, aby [CÍL, který má být splněn], protože [výsledná byznysová HODNOTA pro roli]" doplněná akceptačními kritérii.

Společné autorství uživatelského scénáře může využívat techniky jako je brainstorming nebo tvorba myšlenkových map (mind mapping). Spolupráce umožňuje týmu získat společnou vizi toho, čeho by mělo být dosaženo s přihlédnutím ke třem aspektům (byznys, vývoj, testování).

Správné uživatelské scénáře by měly být tzv. INVEST: nezávislé (Independent), schůdné (Negotiable), hodnotné (Valuable), odhadnutelné (Estimable), malé (Small) a testovatelné (Testable). V určitých situacích se může stát, že zástupce zainteresovaných stran neví, jak otestovat daný uživatelský scénář. To může znamenat, že není dostatečně jasný nebo že neodráží něco, co je pro zainteresované strany cenné, případně že pouze potřebují pomoc při testování (Wake 2003).

4.5.2 Akceptační kritéria

Akceptační kritéria pro uživatelský scénář jsou podmínky, které musí jeho implementace splňovat, aby byla přijata zainteresovanými stranami. Z tohoto hlediska lze na ně pohlížet jako na testovací podmínky, které by měly testy prověřit. Akceptační kritéria jsou obvykle výsledkem konverzace (viz kapitola 4.5.1).

Akceptační kritéria se používají k:

- definování rozsahu uživatelského scénáře,
- dosažení shody mezi zainteresovanými stranami,
- popisu pozitivních i negativních scénářů,
- základní definici akceptačního testování uživatelského scénáře (viz kapitola 4.5.3),
- přesnějšímu plánování a odhadování testování.

Existuje několik způsobů, jak napsat akceptační kritéria pro uživatelský scénář. Dva nejběžnější formáty jsou:

- orientovaný na scénáře (např. formát Given/When/Then používaný v BDD, viz kapitola 2.1.3),
- orientovaný na pravidla (např. verifikační seznam s odrážkami nebo tabulka s mapováním vstupů a výstupů).

Ačkoliv lze většinu akceptačních kritérií dokumentovat v jednom z těchto dvou formátů, může tým použít libovolný jiný (i vlastní) formát za předpokladu, že jsou akceptační kritéria dobře definovaná a jednoznačná.

4.5.3 Vývoj řízený akceptačními testy (ATDD)

ATDD je jedním z přístupů iniciovaných testy (viz kapitola 2.1.3), kdy jsou testovací případy vytvořeny před vlastní implementací uživatelského scénáře členy týmu s různými perspektivami, což jsou např. zákazníci, vývojáři a testeři (Adzic 2009). Testovací případy mohou být prováděny manuálně nebo automatizovaně.

Prvním krokem je obvykle schůzka nad specifikacemi, kde členové týmu analyzují, diskutují a dokumentují uživatelské scénáře a (pokud ještě nejsou definována) jejich akceptační kritéria. Během tohoto procesu jsou obvykle vyřešeny neúplnosti, nejednoznačnosti nebo defekty v uživatelském scénáři.

Dalším krokem je vytvoření testovacích případů, což může být provedeno jak celým týmem společně, tak pouze testery. Testovací případy jsou založeny na akceptačních kritériích a lze je považovat za příklady toho, jak by měl software pracovat, což pomůže týmu správně implementovat uživatelský scénář. Vzhledem k tomu, že příklady a testy jsou vlastně totožné, používají se oba termíny zaměnitelně. Obvykle jsou první testovací případy pozitivní a potvrzují správné chování bez výjimečných situací nebo chybových stavů, často obsahují posloupnost prováděných činností v případech, kdy vše probíhá podle očekávání. Po dokončení pozitivních testovacích případů by měl tým provést negativní testování, a nakonec pokrýt nefunkcionální kvalitativní charakteristiky (např. výkonnostní efektivita nebo použitelnost). Při návrhu testů je možné použít techniky testování popsané v kapitolách 4.2, 4.3 a 4.4.

Testovací případy by měly být vyjádřeny způsobem, který je pro zainteresované strany srozumitelný. Obvykle obsahují věty v přirozeném jazyce zahrnující nezbytné vstupní podmínky (pre-conditions – pokud existují), vstupy a výstupní podmínky (post-conditions). Musí pokrývat všechny charakteristiky uživatelského scénáře a neměly

by přesahovat jeho rámec. Žádné dva testovací případy by neměly popisovat stejné charakteristiky uživatelského scénáře.

Akceptační kritéria mohou podrobně popisovat některé aspekty popsané v uživatelském scénáři. Při zápisu ve formátu podporovaném nějakým frameworkem automatizace testování mohou vývojáři testovací případy automatizovat vytvořením podpůrného kódu již při implementaci užitné vlastnosti popsané uživatelským scénářem. V takovém případě se akceptační testy vlastně stanou spustitelnými požadavky.

5 Management testování – 335 minut

Klíčová slova

analýza rizik, identifikace rizik, management defektů, management rizik, monitoring rizik, monitoring testování, ohodnocení rizik, plán testování, plánování testování, produktové riziko, projektové riziko, přístup k testování, report o defektu, report o postupu prací při testování, riziko, řízení rizik, řízení testování, souhrnný report z testování, testovací kvadranty, testovací pyramida, testování založené na rizicích, úroveň rizika, vstupní kritéria, výstupní kritéria, zmírnění rizik

Studijní cíle

5.1 Plánování testování

FL-5.1.1 (K2) Ilustrovat na příkladech účel a obsah plánu testování.
FL-5.1.2 (K1) Určit, jakou přidanou hodnotou má přítomnost testerů pro plánování iterací a plánování vydání.
FL-5.1.3 (K2) Porovnat a uvést odlišnosti vstupních a výstupních kritérií.
FL-5.1.4 (K3) Použít techniky odhadování pro výpočet pracnosti potřebné při testování.
FL-5.1.5 (K3) Použít prioritizaci testovacích případů.
FL-5.1.6 (K1) Vybavit si koncept testovací pyramidy.
FL-5.1.7 (K2) Shrnout testovací kvadranty a uvědomit si jejich vztah k úrovním testování a typům testů.

5.2 Management rizik

FL-5.2.1 (K1) Identifikovat úroveň rizika pomocí pravděpodobnosti a dopadu rizika.
FL-5.2.2 (K2) Odlišit mezi projektovým a produktovým rizikem.
FL-5.2.3 (K2) Vysvětlit, jak může analýza produktových rizik ovlivnit hloubku a rozsah testování.
FL-5.2.4 (K2) Vysvětlit, jaká opatření lze přijmout v reakci na analyzovaná produktová rizika.

5.3 Monitoring, řízení a dokončení testování

FL-5.3.1 (K1) Vybavit si metriky využívané při testování.
FL-5.3.2 (K2) Shrnout účel a obsah reportů z testování a uvědomit si, pro koho jsou určeny.
FL-5.3.3 (K2) Ilustrovat na příkladech komunikaci stavu testování.

5.4 Konfigurační management

FL-5.4.1 (K2) Shrnout, jak konfigurační management přispívá k testování.

5.5 Management defektů

FL-5.5.1 (K3) Připravit report o defektu.

5.1 Plánování testování

5.1.1 Účel a obsah plánu testování

Plán testování popisuje cíle, zdroje a procesy testování v rámci projektu. Plán testování:

- dokumentuje způsob dosažení cílů testování a harmonogram testování,
- pomáhá zajistit, aby prováděné testovací činnosti splňovaly stanovená kritéria,
- slouží jako způsob komunikace se členy týmu a dalšími zainteresovanými stranami,
- prokazuje, že testování je v souladu se stávající politikou a strategií testování (případně vysvětluje, proč se od nich testování odchyluje).

Plánování testování je průvodcem testerů ve smyslu rozvahy nad budoucími výzvami souvisejícími s riziky, harmonogramy, lidmi, nástroji, náklady, pracností atd. Proces přípravy plánu testování je užitečným nástrojem pro analýzu pracnosti potřebné k dosažení cílů testování stanovených v rámci projektu.

Plán testování obvykle obsahuje:

- kontext testování (např. rozsah, cíle testování, omezení, testovací báze),
- předpoklady a omezení testovacího projektu,
- definici zainteresovaných stran (např. role, odpovědnosti, relevantnost pro testování, potřeby náboru a školení),
- plán komunikace (např. rozdílné způsoby a četnost komunikace, šablony dokumentace),
- registr rizik (např. produktová a projektová rizika),
- přístup k testování (např. úrovně testování, typy testů, techniky testování, výstupy z testování, vstupní a výstupní kritéria, nezávislost testování, požadované metriky, požadavky na testovací data, požadavky na testovací prostředí, odchylky od politiky a strategie testování),
- rozpočet a harmonogram.

Další podrobnosti o plánu testování a jeho obsahu lze nalézt v normě ISO/IEC/IEEE 29119-3.

5.1.2 Přínos testerů při plánování iterací a vydání

V iterativních SDLC se obvykle vyskytují dva druhy plánování: plánování vydání a plánování iterace.

Plánování vydání vyhlíží vpřed směrem k vydání produktu, definuje a upravuje produktový backlog a může zahrnovat rozpracování větších uživatelských scénářů do sady menších. Zároveň slouží jako základ pro definici přístupu k testování a plánu testování napříč všemi iteracemi. Testeři zapojeni do plánování vydání se podílí na specifikaci testovatelných uživatelských scénářů a akceptačních kritérií (viz kapitola 4.5), podílí se na analýzách projektových a produktových rizik (viz kapitola 5.2), provádí odhady pracnosti potřebné k testování uživatelských scénářů (viz kapitola 5.1.4), stanovují přístup k testování a plánují testování související s vydáním.

Plánování iterace vyhlíží směrem ke konci jedné iterace a pracuje s backlogem iterace. Testeři zapojení do plánování iterací se podílí na podrobné analýze rizik uživatelských scénářů, stanovují jejich testovatelnost, podílí se na jejich rozkladu do úkolů (zejména pro testovací činnosti), provádí odhady pracnosti testování pro všechny testovací činnosti a přispívají k identifikaci a zpřesňování funkcionálních a nefunkcionálních aspektů testovaného objektu.

5.1.3 Vstupní kritéria a výstupní kritéria

Vstupní kritéria určují předpoklady pro realizaci dané činnosti. Pokud nejsou splněna vstupní kritéria, bude provedení testovací činnosti pravděpodobně obtížnější, časově náročnější, nákladnější a více rizikové. Mezi typická vstupní kritéria patří dostupnost zdrojů (např. lidé, nástroje, prostředí, testovací data, rozpočet, čas), dostupnost testwaru (např. testovací báze, testovatelné požadavky, uživatelské scénáře, testovací případy) a počáteční úroveň kvality testovaného objektu (např. úspěšnost všech smoke testů).

Výstupní kritéria určují podmínky, kterých musí být dosaženo, aby bylo možné prohlásit činnost za dokončenou. Mezi typická výstupní kritéria patří míra důkladnosti (např. dosažená úroveň pokrytí, počet nevyřešených defektů, hustota defektů, počet neúspěšných testovacích případů) a kritéria dokončení (např. provedení plánovaných testů, provedení statického testování, zaznamenání všech zjištěných defektů, automatizace všech regresních testů).

Za platná výstupní kritéria lze také považovat vyčerpání času nebo finančních prostředků. Pokud v takové situaci (tj. kdy nejsou ostatní výstupní kritéria naplněna) zainteresované strany přezkoumají a přijmou rizika spojená s nasazením produktu do produkce bez dalších testů, lze akceptovat ukončení testování.

V agilním vývoji softwaru se výstupní kritéria často nazývají definice hotového (ang. definition of done) a určují projektovou objektivní metriku pro položky připravené k vydání. Vstupní kritéria, která musí uživatelský scénář splňovat, aby mohly být zahájeny vývojové a/nebo testovací činnosti, se nazývají definice připravenosti.

Vstupní a výstupní kritéria by měla být definována pro každou úroveň testů a mohou se lišit v závislosti na stanovených cílech testování.

5.1.4 Techniky pro odhadování

Odhad pracnosti testování reprezentuje očekávané množství práce, které bude zapotřebí k dosažení cílů testování v projektu. Je důležité všem zainteresovaným stranám objasnit, že odhad vychází z momentálních předpokladů a vždy může být zatížen chybou odhadu, přičemž platí, že odhad pro malé úkoly je obvykle přesnější než pro velké úkoly. Proto je při odhadování vhodné rozložit rozsáhlý úkol na sadu menších a ty následně odhadnout. Tyto učební osnovy popisují čtyři techniky odhadování.

Odhad na základě poměrů. Při této technice založené na metrikách se shromažďují údaje z předchozích projektů v rámci organizace, což umožňuje odvodit „standardizované“ poměrové metriky (vzorce) pro podobné projekty. Takové metriky odvozené z vlastních projektů organizace (např. převzaté z historických dat) jsou obecně nejlepším zdrojem, který lze v procesu odhadování použít a lze je použít k odhadu pracnosti testování nového projektu. Pokud byla například v předchozím projektu pracnost vývoje a pracnost testování v poměru 3:2 a v současném projektu se očekává, že pracnost vývoje bude činit 600 člověko-dnů, lze pracnost testování odhadnout na 400 člověko-dnů.

Extrapolace. Při této technice založené na metrikách se v aktuálním projektu provádí co nejdříve měření s cílem shromažďovat data. S dostatečným počtem pozorování (dat) lze potřebnou zbývající pracnost aproximovat extrapolací těchto dat (obvykle použitím matematického modelu). Tato metoda je velmi vhodná pro iterativní modely SDLC, protože tým může například extrapolovat pracnost testování v nadcházející iteraci jako průměrnou pracnost z posledních tří iterací.

Wideband Delphi. Při této iterativní technice založené na expertech provádějí experti odhady založené na zkušenostech. Každý takový expert (odborník) samostatně odhadne pracnost. Výsledky odhadů jsou shromážděny a pokud se objeví odchylky mimo rozsah dohodnutých hranic, diskutují experti o svých aktuálních odhadech. Každý je poté požádán, aby na základě této zpětné vazby provedl nový odhad, a to opět samostatně. Tento proces se opakuje, dokud není dosaženo shody. Variantou Wideband Delphi, která se běžně používá při agilním vývoji softwaru, je tzv. plánovací poker. Při něm se odhady obvykle provádějí pomocí karet s čísly, která představují rozsah pracnosti.

Tříbodový odhad. Při této technice založené na expertech provádějí experti tři odhady: nejoptimističtější odhad (a), nejpravděpodobnější odhad (m) a nejpesimističtější odhad (b). Výsledný odhad (E) je jejich vážený aritmetický průměr. V nejpoužívanější verzi této techniky se odhad vypočítá jako E = (a + 4 * m + b) / 6. Výhodou této techniky je, že umožňuje expertům vypočítat chybu měření, obvykle ve formě směrodatné odchylky: SD = (b - a) / 6. Pokud jsou např. odhady (v člověko-hodinách): a = 6, m = 9 a b = 18, pak výsledný odhad je 10 ± 2 člověko-hodin (tj. mezi 8 a 12 člověko-hodinami), protože E = (6 + 4 * 9 + 18) / 6 = 10 a SD = (18 - 6) / 6 = 2.

Více informací o těchto a mnoha dalších technikách odhadování pracnosti testování lze nalézt v Kan 2003, Koomen 2006 a Westfall 2009.

5.1.5 Prioritizace testovacích případů

Jakmile jsou testovací případy a testovací procedury vytvořeny a sestaveny do testovacích sad, lze tyto testovací sady řadit do harmonogramu provádění definující pořadí spouštění.

Při stanovení priorit testovacích případů lze zohlednit různé faktory. Nejčastěji používané strategie prioritizace testovacích případů jsou následující:

- **Prioritizace na základě rizik**, kdy pořadí provádění testů vychází z výsledků analýzy rizik (viz kapitola 5.2.3). Nejprve se provedou testovací případy pokrývající nejdůležitější rizika.
- **Prioritizace na základě pokrytí**, kdy je pořadí provádění testů založeno na určitém pokrytí (např. pokrytí příkazů). Testovací případy dosahující nejvyššího pokrytí jsou provedeny jako první. V jiné variantě (nazývané prioritizace dodatečného pokrytí) se nejprve provede testovací případ s nejvyšším pokrytím. Každý následující testovací případ je ten, který dosáhne nejvyššího dodatečného pokrytí.
- **Prioritizace na základě požadavků**, kdy pořadí provádění testů vychází z priorit požadavků trasovatelných zpět k odpovídajícím testovacím případům. Priority požadavků definují zainteresované strany, testovací případy související s nejdůležitějšími požadavky jsou prováděny jako první.

V ideálním případě by měly být testovací případy seřazeny k provádění na základě úrovně jejich priority, například pomocí jedné z výše uvedených strategií prioritizace. Tento postup však nemusí fungovat, pokud mezi testovacími případy nebo testovanými užitnými vlastnostmi existují závislosti. Pokud testovací případ s vyšší prioritou závisí na testovacím případu s nižší prioritou, musí být nejdříve proveden testovací případ s nižší prioritou.

Pořadí provádění testů musí také zohledňovat dostupnost zdrojů, např. požadovaných testovacích nástrojů, testovacího prostředí nebo osob, které mohou být k dispozici pouze po určitou dobu.

5.1.6 Testovací pyramida

Testovací pyramida je model, který ukazuje, že různé testy mohou mít různou granularitu. Představuje pomůcku pro určení míry automatizovaných testů tím, že týmu názorně ukazuje rozložení pracnosti v jednotlivých úrovních (typech) automatizace potřebné k dosažení specifických cílů každé úrovně.

Vrstvy pyramidy představují skupiny testů. Čím vyšší vrstva, tím nižší granularita testu, menší izolace testu a delší doba provádění testu. Testy ve spodní vrstvě jsou malé, izolované, rychlé a ověřují malou část funkcionality, takže k dosažení rozumného pokrytí je jich obvykle potřeba velké množství. Horní vrstva představuje komplexní vysokoúrovňové E2E (end-to-end) testy. Ty jsou obecně pomalejší než testy z nižších vrstev a obvykle ověřují velkou část funkcionality, takže pro dosažení rozumného pokrytí je jich obvykle zapotřebí jen několik. Počet a pojmenování jednotlivých vrstev se může lišit, například původní model testovací pyramidy (Cohn 2009) definuje tři vrstvy (testy komponent, testy služeb a testy uživatelského rozhraní). Jiné modely definují jednotkové testy (testy komponent), integrační testy komponent a end-to-end testy, obecně lze použít i jiné úrovně testování (viz kapitola 2.2.1).

5.1.7 Testovací kvadranty

Testovací kvadranty definované Brianem Marickem (Marick 2003, Crispin 2008), dávají do souvislosti úrovně testování s příslušnými typy testů, činnostmi, technikami testování a pracovními produkty v agilním vývoji softwaru. Model je pomůckou pro management testování při vizualizaci těchto vztahů s cílem zajišťovat, že všechny vhodné typy a úrovně testů jsou zahrnuty do SDLC, a pro pochopení toho, že některé typy testů jsou pro určité úrovně testů relevantnější než jiné. Poskytuje způsob, jak od sebe rozlišit a popsat typy testů všem zainteresovaným stranám včetně vývojářů, testerů a zástupců byznysu.

V tomto modelu osa Y rozlišuje testy zaměřené na byznys nebo na technologii a osa X pak testy zaměřené na podporu týmu (tj. ty, které pomáhají usměrňovat vývojové aktivity) nebo kritiku produktu (tj. takové, které pomáhají měřit jeho chování proti očekáváním). Kombinace těchto dvou pohledů (os) určuje čtyři kvadranty:

- **Kvadrant Q1 (zaměřené na technologii a podporu týmu)**. Tento kvadrant obsahuje testy komponent a integrační testy komponent. Tyto testy by měly být automatizovány a zahrnuty do procesu CI.

- **Kvadrant Q2 (zaměřené na byznys a podporu týmu)**. Tento kvadrant obsahuje funkcionální testy, příklady, testy uživatelských scénářů, UX prototypy, testování rozhraní (API) a simulace. Tyto testy ověřují akceptační kritéria a mohou být manuální i automatizované.
- **Kvadrant Q3 (zaměřené na byznys a kritiku produktu)**. Tento kvadrant obsahuje průzkumné testování, testování použitelnosti a uživatelské akceptační testování. Tyto testy jsou uživatelsky orientované a často manuální.
- **Kvadrant Q4 (zaměřené na technologii a kritiku produktu)**. Tento kvadrant obsahuje smoke testy a nefunkcionální testy (kromě testů použitelnosti). Tyto testy jsou často automatizované.

5.2 Management rizik

Organizace čelí mnoha interním a externím faktorům přinášejícím nejistotu v tom, zda a kdy dosáhnou svých cílů (ISO 31000). Management rizik jim umožňuje zvýšit pravděpodobnost dosažení těchto cílů, zlepšit kvalitu jejich produktů a zvýšit důvěru všech zainteresovaných stran.

Hlavními činnostmi v oblasti managementu rizik jsou:

- **analýza rizik** (obsahuje identifikaci a ohodnocení rizik, viz kapitola 5.2.3),
- **řízení rizik** (obsahuje zmírňování a monitoring rizik, viz kapitola 5.2.4).

Přístup k testování, při kterém jsou testovací činnosti vybírány, prioritizovány a spravovány na základě analýzy rizik společně s řízením rizik, se nazývá testování založené na rizicích.

5.2.1 Definice rizika a jeho atributy

Riziko je možná událost, nebezpečí, hrozba nebo situace, jejíž výskyt má nepříznivý vliv. Riziko lze charakterizovat dvěma faktory:

- pravděpodobnost rizika – pravděpodobnost vzniku rizika vyjádřená v procentech nebo v intervalu (0;1),
- dopad rizika (škoda) – důsledky výskytu takové události.

Tyto dva faktory vyjadřují úroveň rizika, která je jeho mírou (metrikou). Čím vyšší je úroveň rizika, tím důležitější je jeho ošetření.

5.2.2 Projektová a produktová rizika

Při testování softwaru se obecně zabýváme dvěma typy rizik – projektová rizika a produktová rizika.

Projektová rizika se týkají managementu a řízení projektu. Mezi projektová rizika patří:

- organizační problémy (např. zpoždění při dodávání pracovních produktů, nepřesné odhady, snižování nákladů),
- problémy s lidmi (např. nedostatečné dovednosti, konflikty, komunikační problémy, nedostatek zaměstnanců),
- technické problémy (např. narůstající rozsah, špatná podpora nástrojů),
- problémy s dodavateli (např. selhání dodávky třetí strany, úpadek dodavatele).

Projektová rizika mohou mít dopad na harmonogram, rozpočet nebo rozsah projektu, což ovlivňuje schopnost projektu dosáhnout svých cílů.

Produktová rizika souvisejí s kvalitativními charakteristikami produktu (např. popsanými v modelu kvality ISO 25010). Mezi příklady produktových rizik patří:

- chybějící nebo nesprávná funkcionalita,
- nesprávné výpočty,
- chyby při běhu (runtime errors),
- špatná architektura,
- neefektivní algoritmy,
- nedostatečná doba odezvy,
- špatná uživatelská zkušenost,

- bezpečnostní zranitelnosti.

Produktová rizika mohou mít různé negativní důsledky, například:

- nespokojenost uživatelů,
- ztráta příjmů, důvěry, pověsti,
- škody způsobené třetím stranám,
- vysoké náklady na údržbu,
- přetížení helpdesku,
- trestněprávní postihy,
- fyzické poškození, zranění, v extrémních případech smrt.

5.2.3 Analýza produktových rizik

Z hlediska testování je cílem analýzy produktových rizik poskytnout povědomí o produktových rizicích s cílem nasměrovat pracnost při testování tak, aby došlo k minimalizaci jejich reziduálních úrovní. V ideálním případě začíná analýza produktových rizik v rané fázi SDLC. Analýza produktových rizik se skládá z identifikace a ohodnocení rizik.

Identifikace rizik spočívá ve vytvoření komplexního seznamu rizik. Všechny zainteresované strany mohou identifikovat rizika pomocí různých technik a nástrojů, např. brainstormingu, workshopů, rozhovorů nebo grafů příčin a následků.

Ohodnocení rizik zahrnuje kategorizaci identifikovaných rizik, určení jejich pravděpodobnosti, dopadu a úrovně, stanovení priorit a navržení způsobů jejich řešení. Kategorizace pomáhá při přiřazování zmírňujících opatření, protože obvykle lze rizika spadající do stejné kategorie zmírnit podobnými činnostmi.

Ohodnocení rizik může využívat kvantitativní nebo kvalitativní přístup (příp. obojí). Při kvantitativním přístupu se úroveň rizika vypočítá jako násobek pravděpodobnosti rizika a dopadu rizika. Při kvalitativním přístupu lze úroveň rizika stanovit pomocí matice rizik.

Analýza produktových rizik může ovlivnit důkladnost a rozsah testování. Její výsledky se používají k:

- určení rozsahu testování, které má být provedeno,
- určení konkrétních úrovní testování a návrhu typů testů, které mají být provedeny,
- určení technik testování, které mají být použity,
- určení pokrytí, kterého má být dosaženo,
- odhadu pracnosti testování vyžadovaného pro každý úkol,
- stanovení priorit testování s cílem co nejrychlejšího nalézání kritických defektů,
- určení, zda by mohly být ke snížení rizika použity i jiné činnosti (kromě testování).

5.2.4 Řízení produktových rizik

Řízení produktových rizik zahrnuje všechna opatření, která jsou přijata v reakci na identifikovaná a ohodnocená produktová rizika. Skládá se ze zmírňování rizik a monitoringu rizik.

Obsahem **zmírňování rizik** je zavedení opatření navržených při ohodnocení rizik s cílem snížit úroveň rizika. Jakmile je riziko analyzováno, je možné na něj reagovat několika způsoby, např. jeho zmírněním pomocí testování, přijetím, přenosem nebo záložním plánem (Veenendaal 2012). Opatření, která lze přijmout ke zmírnění produktových rizik prostřednictvím testování, jsou následující:

- výběr testerů se zkušenostmi a dovednostmi vhodnými pro daný typ rizika,
- použití vhodné úrovně nezávislosti testování,
- provedení revizí a statické analýzy,
- použití vhodných technik testování a úrovní pokrytí,
- použití vhodných typů testů zaměřených na dotčené kvalitativní charakteristiky,
- provedení dynamického testování (včetně regresního testování).

Cílem **monitoringu rizik** je zajišťování efektivity zmírňujících opatření, zisk dalších informací pro zlepšení ohodnocení rizik a identifikace nově vznikajících rizik.

5.3 Monitoring, řízení a dokončení testování

Monitoring testování se zabývá shromažďováním informací o testování. Tyto informace se využívají k posouzení postupu prací při testování a ke stanovení, zda jsou naplněna výstupní kritéria testování nebo dokončeny testovací úkoly spojené s výstupními kritérii (jako např. splnění cílů pro pokrytí produktových rizik, požadavků nebo akceptačních kritérií).

Řízení testování využívá informace z monitoringu testování k tomu, aby byly (ve formě nařízení nebo směrnic) poskytnuty pokyny a nezbytná nápravná opatření k dosažení co nejúčinnějšího a nejefektivnějšího testování. Mezi příklady takových nařízení patří:

- přehodnocení priorit testů při výskytu identifikovaného rizika,
- přehodnocení, zda v důsledku změn (přepracování) splňuje daná položka testování vstupní nebo výstupní kritéria,
- úprava harmonogramu testování s ohledem na zpoždění v dodání testovacího prostředí,
- přidávání nových zdrojů tam, kde je potřeba a kdy je potřeba.

V rámci **dokončení testování** se shromažďují data z dokončených testovacích činností za účelem konsolidace zkušeností, testwaru a dalších důležitých informací. K činnostem souvisejícím s dokončením testování dochází v rámci projektových milníků jako je dokončení úrovně testování, dokončení iterace při agilním vývoji, dokončení (nebo zrušení) testovacího projektu, vydání softwarového systému nebo dokončení servisního vydání (maintenance release).

5.3.1 Metriky používané v testování

Testovací metriky jsou shromažďovány s cílem ukazovat postup proti plánovanému harmonogramu a rozpočtu, aktuální kvalitu testovaného objektu a efektivitu testovacích činností s ohledem na cíle projektu nebo iterace. Monitoring testování shromažďuje různé metriky pro podporu řízení a dokončení testování.

Mezi typické testovací metriky patří:

- metriky o postupu projektu (např. dokončení úkolu, využití zdrojů, pracnost testování),
- metriky o postupu testování (např. postup implementace testovacích případů, postup přípravy testovacích prostředí, počet spuštěných / nespuštěných testovacích případů, počet provedených testů, které prošly / selhaly, doba provedení testu),
- metriky týkající se kvality produktu (např. dostupnost, doba odezvy, střední doba mezi poruchami),
- metriky nad defekty (např. počet a priority zjištěných / opravených defektů, hustota defektů, procento zjištěných defektů – DDP, defect detection percentage),
- metriky nad riziky (např. úroveň reziduálních rizik),
- metriky pokrytí (např. pokrytí požadavků, pokrytí kódu),
- metriky týkající se nákladů (např. náklady na testování, náklady na kvalitu v rámci organizace).

5.3.2 Účel, obsah a cílové skupiny reportů z testování

Cílem reportování testů je shrnutí a komunikace informací z testování při jeho průběhu a po něm. Reporty o postupu prací při testování podporují průběžné řízení testování. Musí poskytovat dostatek informací pro provedení změn v harmonogramu testování, zdrojích nebo plánu testování, pokud jsou tyto změny nutné z důvodu odchylky od plánu nebo změny okolností. Souhrnné reporty z testování shrnují určitou fázi testování (např. úroveň testování, testovací cyklus, iteraci) a mohou poskytnout informace pro následné testování.

Během monitoringu a řízení testování vytváří testovací tým reporty o postupu prací při testování s cílem poskytovat zainteresovaným stranám informace. Reporty o postupu prací při testování jsou obvykle generovány pravidelně (např. denně, týdně atd.) a obsahují:

- testovací období,
- postup prací při testování (např. náskok nebo zpoždění oproti harmonogramu) včetně všech významných odchylek,
- překážky při testování a jejich řešení,

- testovací metriky (viz kapitola 5.3.1),
- nová a změněná rizika během testovacího období,
- testování plánované pro následující období.

Souhrnný report z testování se připravuje v průběhu fáze dokončení testování, kdy je projekt, úroveň testování nebo typ testu ukončen, a kdy jsou (v ideálním případě) splněna stanovená výstupní kritéria. Tento report využívá informace z dílčích reportů o postupu prací při testování a doplňuje je o další data. Typické souhrnné reporty z testování obsahují:

- shrnutí testů,
- vyhodnocení testování a kvality produktu na základě původního plánu testování (tj. cílů testování a výstupních kritérií),
- odchylky od plánu testování (např. rozdíly oproti plánovanému harmonogramu, trvání a pracnosti),
- překážky při testování a jejich řešení,
- testovací metriky na základě reportů o postupu prací při testování,
- neošetřená rizika, nevyřešené defekty,
- ponaučení (lessons learned), které jsou relevantní pro testování.

Různé cílové skupiny potřebují v těchto reportech různé informace a ovlivňují tím míru formálnosti a četnost reportování. Podávání zpráv o postupu prací při testování ostatním členům stejného týmu je většinou časté a neformální, zatímco podávání souhrnných reportů z testování z ukončeného projektu se provádí typicky pouze jednou, a to dle definované šablony.

V normě ISO/IEC/IEEE 29119-3 lze nalézt šablony a příklady reportů o postupu prací při testování (nazývané reporty o stavu testů) a souhrnných reportů z testování.

5.3.3 Komunikování stavu testování

Optimální způsob komunikování stavu testování se liší v závislosti na potřebách managementu testování, strategiích testování v organizaci, regulatorních normách nebo v případě samoorganizujících se týmů (viz kapitola 1.5.2) na samotném týmu. Mezi možnosti komunikace patří:

- verbální komunikace se členy týmu a dalšími zainteresovanými stranami,
- dashboardy (např. CI/CD dashboard, tabule s úkoly a grafy burn-down),
- elektronické komunikační kanály (např. e-mail, chat),
- online dokumentace,
- formální reporty z testování (viz kapitola 5.3.2).

Lze použít jednu nebo kombinaci několika možností, formálnější komunikace může být vhodnější pro distribuované týmy, kde není přímá osobní komunikace vždy možná kvůli geografické vzdálenosti nebo časovým rozdílům. Obvykle se různé zainteresované strany zajímají o jiný typ informací, proto by měla být komunikace odpovídajícím způsobem přizpůsobena.

5.4 Konfigurační management

V oblasti testování je konfigurační management disciplína, která slouží pro identifikaci, řízení a sledování pracovních produktů. Konfiguračními položkami mohou být jakékoliv plány testování, strategie testování, testovací podmínky, testovací případy, skripty, výsledky testů, protokoly testů (test logs) a reporty z testování.

V případě komplexní konfigurační položky (např. testovací prostředí) lze v rámci konfiguračního managementu zaznamenávat dílčí položky, ze kterých se tato komplexní položka skládá, jejich vztahy a verze. V momentě, kdy je konfigurační položka schválena pro testování, stává se součástí tzv. baseline a lze ji měnit pouze prostřednictvím formálního procesu pro řízení změn.

V případě vytvoření nové baseline je úkolem konfiguračního managementu uchovávat záznamy o všech změněných konfiguračních položkách. Pokud je například nutné reprodukovat předchozí výsledky testů, je možné vrátit se k předchozí baseline.

Konfigurační management jako podpůrná disciplína testování zajišťuje následující:

- Všechny konfigurační položky včetně položek testování (jednotlivých částí testovaného objektu) jsou jednoznačně identifikovány, jejich verze jsou řízeny, všechny změny jsou evidovány a vztahy s jinými konfiguračními položkami jsou zaznamenány tak, aby bylo možné zajistit trasovatelnost v průběhu celého procesu testování.
- V dokumentaci testování vedou na všechny identifikované dokumenty a softwarové položky jednoznačné odkazy.

Průběžná integrace, průběžné dodávání, průběžné nasazování a s těmito procesy související testování jsou obvykle implementovány jako součást automatizované DevOps pipeline (viz kapitola 2.1.4), jejíž součástí je obvykle i automatizovaný konfigurační management.

5.5 Management defektů

Vzhledem k tomu, že jedním z hlavních cílů testování je nalezení defektů, je zavedení procesu managementu defektů nezbytné. Tento proces musí dodržovat všechny zainteresované strany a zahrnuje minimálně definici pracovního postupu (workflow) pro zpracování jednotlivých anomálií od jejich odhalení až po jejich uzavření a pravidla pro jejich klasifikaci. Pracovní postup obvykle zahrnuje činnosti, jejichž cílem je zaznamenávat nahlášené anomálie, analyzovat je a klasifikovat, rozhodnout o vhodné reakci (typicky opravit nebo ponechat) a uzavření reportu o defektu. Anomálie mohou být hlášeny v kterékoli fázi SDLC a jejich forma často závisí na použitém SDLC.

Podobným způsobem je vhodné řešit i defekty zjištěné statickým testováním (zejména statickou analýzou).

Platí, že ne všechny nahlášené anomálie musí být nutně skutečné defekty (např. falešně pozitivní výsledky testů nebo změnové požadavky). Toto vyhodnocení je prováděno v rámci zpracování reportu o defektu. Reporty o defektu mají obvykle následující cíle:

- Poskytovat osobám odpovědným za zpracování a řešení hlášených defektů dostatečné informace pro vyřešení problému.
- Poskytovat způsob sledování kvality pracovního produktu.
- Navrhovat nápady pro zlepšování procesů vývoje a procesů testování.

Report o defektu zaznamenaný během dynamického testování obvykle obsahuje:

- jedinečný identifikátor,
- název obsahující krátké shrnutí reportované anomálie,
- datum zjištění anomálie, reportující organizace a autor (včetně role autora),
- identifikaci testovaného objektu a testovacího prostředí,
- kontext defektu (např. provedený testovací případ, provedená testovací činnost, fáze SDLC a další relevantní informace jako např. použitá technika testování, použitý kontrolní seznam nebo použitá testovací data),
- popis selhání umožňující reprodukci a vyřešení včetně kroků, které anomálii vyvolaly, všech relevantních protokolů testů (test logs), záloh databází, snímků obrazovky nebo videozáznamů,
- očekávané a skutečné výsledky,
- závažnost defektu (míru dopadu) na zájmy zainteresovaných stran nebo na požadavky,
- naléhavost / prioritu opravy,
- stav defektu (např. otevřený, odložený, duplicitní, čekající na opravu, čekající na konfirmační testování, znovu otevřený, uzavřený, zamítnutý),
- odkazy (např. na testovací případ).

Některé z těchto informací mohou být automaticky vkládány nástrojem pro management defektů (např. identifikátor, datum, autor a počáteční stav). Šablony dokumentů pro report o defektu a příklady reportů o defektu lze nalézt v normě ISO/IEC/IEEE 29119-3 (norma však používá termín zpráva o incidentu).

6 Testovací nástroje – 20 minut

Klíčová slova

automatizace testů

Studijní cíle

6.1 Nástroje pro podporu testování

FL-6.1.1 (K2) Vysvětlit, jak různé typy testovacích nástrojů podporují testování.

6.2 Výhody a rizika automatizace testů

FL-6.2.1 (K1) Vybavit si výhody a rizika automatizace testů.

6.1 Nástroje pro podporu testování

Testovací nástroje podporují a usnadňují mnoho testovacích aktivit. Mezi takové nástroje patří například:

- nástroje pro management testování – zvyšují efektivitu testování usnadněním správy SDLC, požadavků, testů, defektů, konfigurací,
- nástroje pro statické testování – podporují provádění revizí a statické analýzy,
- nástroje pro návrh a implementaci testů – usnadňují vytváření testovacích případů, testovacích dat a testovacích procedur,
- nástroje pro provádění testů a měření jejich pokrytí – usnadňují automatizované provádění testů a měření pokrytí,
- nástroje pro nefunkcionální testování – umožňují provádět nefunkcionální testování, které je obtížné nebo nemožné provést manuálně,
- nástroje DevOps – podporují pipeline v DevOps, sledování pracovních toků, procesy automatizovaného sestavování, CI/CD,
- nástroje pro spolupráci – usnadňují komunikaci,
- nástroje podporující škálovatelnost a standardizaci nasazení (např. virtuální počítače, nástroje pro práci s kontejnery),
- jakýkoli jiný nástroj, který pomáhá při testování (např. tabulkový procesor může být v kontextu testování považován za testovací nástroj).

6.2 Výhody a rizika automatizace testů

Prosté pořízení nástroje není zárukou úspěchu. Každý nový nástroj vyžaduje určitou pracnost pro dosažení reálných a trvalých přínosů (např. na zavedení nástroje, jeho údržbu a školení). Existují také určitá rizika, u kterých je vhodné provést analýzu a zmírnění jejich dopadu.

Mezi možné výhody použití automatizace testů patří:

- Úspora času snížením množství opakované manuální práce (např. provádění regresních testů, opětovné zadávání stejných testovacích dat, porovnávání očekávaných výsledků se skutečnými nebo kontroly kódu dle daného standardu programování).
- Předcházení lidských chyb díky vyšší konzistenci a opakovatelnosti (např. testy jsou důsledně odvozovány z požadavků, testovací data jsou vytvářena systematicky nebo testy jsou prováděny nástrojem ve stále stejném pořadí se stejnou frekvencí).
- Objektivnější posouzení (např. pokrytí) a provádění opatření, která jsou pro člověka příliš komplikovaná na odvození.
- Snazší přístup k informacím o testování pro podporu managementu testování a reportování testů (např. statistiky, grafy a souhrnná data o průběhu testů, míře defektů a délce provádění testů).
- Zkrácení doby provádění testů vedoucí k dřívější detekci defektů, rychlejší zpětné vazbě a rychlejšímu uvedení na trh.
- Úspora času testerů, který je možno využít na návrh nových, důkladnějších a efektivnějších testů.

Mezi možná rizika použití automatizace testů patří:

- Nerealistická očekávání od daného nástroje (včetně funkcionality a snadnosti použití).
- Nepřesné odhady času, nákladů, úsilí potřebného k zavedení nástroje, udržování testovacích skriptů a změně stávajícího procesu manuálního testování.
- Používání nástroje v situacích, kdy je manuální testování vhodnější.
- Přílišná spoléhání se na nástroj, např. ve smyslu ignorování potřeby kritického myšlení člověka.
- Závislost na dodavateli nástroje, který může ukončit činnost, přestat vyvíjet nástroj, prodat nástroj jinému dodavateli nebo poskytovat nedostatečnou podporu (např. reakce na dotazy, upgrady a opravy defektů).
- Používání open-source nástroje, jehož vývoj může být zastaven nebo jehož vnitřní komponenty mohou vyžadovat poměrně časté aktualizace a další vývoj.

- Nekompatibilita technologie automatizace testování s technologiemi pro vývoj.
- Výběr nevhodného nástroje, který nesplňuje regulatorní požadavky a/nebo bezpečnostní normy.

7 Reference

Normy

- ISO/IEC/IEEE 29119-1 (2022) Softwarové a systémové inženýrství – Testování softwaru – Část 1: Obecné pojmy
- ISO/IEC/IEEE 29119-2 (2021) Softwarové a systémové inženýrství – Testování softwaru – Část 2: Procesy testování
- ISO/IEC/IEEE 29119-3 (2021) Softwarové a systémové inženýrství – Testování softwaru – Část 3: Dokumentace testování
- ISO/IEC/IEEE 29119-4 (2021) Softwarové a systémové inženýrství – Testování softwaru – Část 4: Techniky testování
- ISO/IEC 25010 (2011) Systémové a softwarové inženýrství – Požadavky a hodnocení kvality systémů a softwaru (SQuaRE), Modely kvality systémů a softwaru
- ISO/IEC 20246 (2017) Softwarové a systémové inženýrství – revize pracovních produktů
- ISO/IEC/IEEE 14764:2022 – Softwarové inženýrství – Procesy životního cyklu softwaru – Údržba
- ČSN ISO 31000 (2018) Management rizik – Principy a směrnice

Knihy

- Adzic, G. (2009) Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri Limited
- Ammann, P. a Offutt, J. (2016) Introduction to Software Testing (2e), Cambridge University Press
- Andrews, M. a Whittaker, J. (2006) How to Break Web Software: Functional and Security Testing of Web Applications and Web Services, Addison-Wesley Professional
- Beck, K. (2003) Test Driven Development: By Example, Addison-Wesley
- Beizer, B. (1990) Software Testing Techniques (2e), Van Nostrand Reinhold: Boston MA
- Boehm, B. (1981) Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ
- Buxton, J.N. a Randell B., eds (1970), Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27–31 October 1969, s. 16
- Chelimsky, D. et al. (2010) The Rspec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends, The Pragmatic Bookshelf: Raleigh, NC
- Cohn, M. (2009) Succeeding with Agile: Software Development Using Scrum, Addison-Wesley
- Copeland, L. (2004) A Practitioner's Guide to Software Test Design, Artech House: Norwood MA
- Craig, R. a Jaskiel, S. (2002) Systematic Software Testing, Artech House: Norwood MA
- Crispin, L. a Gregory, J. (2008) Agile Testing: A Practical Guide for Testers and Agile Teams, Pearson Education: Boston MA
- Forgács, I., a Kovács, A. (2019) Practical Test Design: Selection of traditional and automated test design techniques, BCS, The Chartered Institute for IT
- Gawande A. (2009) The Checklist Manifesto: How to Get Things Right, New York, NY: Metropolitan Books
- Gärtner, M. (2011), ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, Pearson Education: Boston MA
- Gilb, T., Graham, D. (1993) Software Inspection, Addison Wesley:
- Hendrickson, E. (2013) Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing, The Pragmatic Programmers
- Hetzel, B. (1988) The Complete Guide to Software Testing, 2. vydání, John Wiley and Sons
- Jeffries, R., Anderson, A., Hendrickson, C. (2000) Extreme Programming Installed, Addison-Wesley Professional
- Jorgensen, P. (2014) Software Testing, A Craftsman's Approach (4e), CRC Press: Boca Raton FL
- Kan, S. (2003) Metrics and Models in Software Quality Engineering, 2. vydání, Addison-Wesley
- Kaner, C., Falk, J., a Nguyen, H.Q. (1999) Testing Computer Software, 2. vydání, Wiley

- Kaner, C., Bach, J. a Pettichord, B. (2011) Lessons Learned in Software Testing: A Context-Driven Approach, 1. vydání, Wiley
- Kim, G., Humble, J., Debois, P. a Willis, J. (2016) The DevOps Handbook, Portland, OR
- Koomen, T., van der Aalst, L., Broekman, B. a Vroon, M. (2006) TMap Next for result-driven testing, UTN Publishers, Nizozemsko
- Myers, G. (2011) The Art of Software Testing, (3e), John Wiley & Sons: New York NY
- O'Regan, G. (2019) Concise Guide to Software Testing, Springer Nature Switzerland
- Pressman, R.S. (2019) Software Engineering. A Practitioner's Approach, 9. vydání, McGraw Hill
- Roman, A. (2018) Thinking-Driven Testing. The Most Reasonable Approach to Quality Control, Springer Nature Switzerland
- Van Veenendaal, E (ed.) (2012) Practical Risk-Based Testing, The PRISMA Approach, UTN Publishers: Nizozemsko
- Watson, A.H., Wallace, D.R. a McCabe, T.J. (1996) Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, U.S. Dept. of Commerce, Technology Administration, NIST
- Westfall, L. (2009) The Certified Software Quality Engineer Handbook, ASQ Quality Press
- Whittaker, J. (2002) How to Break Web Software: A Practical Guide to Testing, Pearson
- Whittaker, J. (2009) Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, Addison Wesley
- Whittaker, J. a Thompson, H. (2003) How to Break Software Security, Addison Wesley
- Wiegers, K. (2001) Peer Reviews in Software: A Practical Guide, Addison-Wesley Professional

Články a webové stránky

- Brykczynski, B. (1999) "A survey of software inspection checklists," ACM SIGSOFT Software Engineering Notes, 24(1), s. 82-89
- Enders, A. (1975) "An Analysis of Errors and Their Causes in System Programs," IEEE Transactions on Software Engineering 1(2), s. 140-149
- Manna, Z., Waldinger, R. (1978) "The logic of computer programming," IEEE Transactions on Software Engineering 4(3), s. 199-229
- Marick, B. (2003) Exploration through Example, http://www.exampler.com/old-blog/2003/08/21.1.html#agile-testing-project-1
- Nielsen, J. (1994) "Enhancing the explanatory power of usability heuristics", Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, ACM Press, s. 152–158
- Salman. I. (2016) "Cognitive biases in software quality and testing", Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16), ACM, s. 823-826.
- Wake, B. (2003) "INVEST in Good Stories, and SMART Tasks", https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

8 Příloha A – Studijní cíle a kognitivní úroveň znalostí

Pro účely těchto učebních osnov jsou definovány studijní cíle. Každé téma v osnovách může být přezkoušeno dle odpovídajícího studijního cíle. Studijní cíle začínají vždy slovesem, které odpovídá jeho kognitivní úrovni znalostí, viz níže.

Úroveň 1: Zapamatovat si (K1) – kandidát je schopen zapamatovat si, určit nebo vybavit si daný termín nebo koncept.

Slovesa: identifikovat, vybavit si, zapamatovat si, určit.

Příklady:

- „Identifikovat typické cíle testování."
- „Vybavit si koncept testovací pyramidy."
- „Určit, jakou přidanou hodnotou má přítomnost testerů pro plánování iterací a plánování vydání."

Úroveň 2: Porozumět (K2) – Kandidát dokáže označit příčiny nebo vysvětlení k výrokům vztahující se k tématu, dokáže shrnout, porovnat, kategorizovat a uvést příklady pro daný koncept testování.

Slovesa: kategorizovat, porovnat, uvést odlišnosti, rozlišit, odlišit, ilustrovat na příkladech, vysvětlit, uvést příklady, interpretovat, shrnout.

Příklady:

- „Kategorizovat různé možnosti pro psaní akceptačních kritérií."
- „Porovnat různé role v testování" (hledejte podobnosti a/nebo rozdíly).
- „Odlišit projektová rizika od produktových rizik" (umožňuje diferencovat pojmy).
- „Ilustrovat na příkladech účel a obsah plánu testování."
- „Vysvětlit dopad kontextu na proces testování."
- „Shrnout aktivity procesu revizí."

Úroveň 3: Použít (K3) – kandidát je schopen aplikovat daný postup, pokud je konfrontován s obdobným úkolem, nebo zvolit správný postup a aplikovat jej na daný kontext.

Slovesa: aplikovat, implementovat, připravit, použít.

Příklady:

- „Aplikovat prioritizaci testovacího případu" (mělo by odkazovat na postup, techniku, proces, algoritmus atd.).
- „Připravit report o defektu."
- „Použít techniku analýzy hraničních hodnot k odvození testovacích případů."

Reference (pro kognitivní úrovně studijních cílů)

- Anderson, L. W. a Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon

9 Příloha B – Trasovací matice profesních a studijních cílů

Tato část dává do souvislosti studijní a profesní cíle těchto učebních osnov ve formě trasovací matice.

Profesní cíle: Učební osnovy pro základní stupeň		FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
BO1	Chápat, co je testování a proč je prospěšné.	6													
BO2	Chápat základní koncepty testování softwaru.		22												
BO3	Určit přístup k testování a činnosti, které mají být provedeny v závislosti na kontextu.			6											
BO4	Posoudit a zlepšit kvalitu dokumentace.				9										
BO5	Zvýšit efektivitu a účinnost testování.					20									
BO6	Sladit proces testování s životním cyklem vývoje softwaru (SDLC).						6								
BO7	Rozumět principům managementu testování.							6							
BO8	Psát jasné a srozumitelné reporty o defektu a umět je komunikovat.								1						
BO9	Chápat faktory, které ovlivňují priority a pracnost činností souvisejících s testováním.									7					
BO10	Pracovat jako součást multifunkčního týmu.										8				
BO11	Rozumět rizikům a přínosům automatizace testů.											1			
BO12	Identifikovat základní dovednosti potřebné pro testování.												5		
BO13	Chápat vliv rizika na testování.													4	
BO14	Efektivně reportovat o průběhu a kvalitě testování.														4

Kapitola/ podkapitola/ část	Studijní cíl	K-úroveň	PROFESNÍ CÍLE													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
Kapitola 1	**Základy testování**															
1.1	**Co je testování?**															
1.1.1	Identifikovat typické cíle testování.	K1	X													
1.1.2	Odlišit testování od ladění.	K2		X												
1.2	**Proč je testování nezbytné?**															
1.2.1	Ilustrovat na příkladech, proč je testování nezbytné.	K2	X													
1.2.2	Vybavit si vztah mezi testováním a zajištěním kvality.	K1		X												
1.2.3	Rozlišit mezi kořenovou příčinou, chybou, defektem a selháním.	K2		X												
1.3	**Principy testování**															
1.3.1	Vysvětlit sedm principů testování.	K2		X												
1.4	**Testovací činnosti, testware a role v testování**															
1.4.1	Shrnout různé testovací činnosti a úkoly.	K2			X											
1.4.2	Vysvětlit vliv kontextu na proces testování.	K2			X			X								
1.4.3	Rozlišit mezi různými druhy testwaru	K2			X											

	podporujícími testovací činnosti.															
1.4.4	Vysvětlit hodnotu udržování trasovatelnosti.	K2				X	X									
1.4.5	Porovnat různé role v testování.	K2										X				
1.5	**Základní dovednosti a osvědčené postupy v testování**															
1.5.1	Uvést příklady obecných dovedností požadovaných pro testování.	K2												X		
1.5.2	Vybavit si výhody týmového přístupu.	K1										X				
1.5.3	Uvědomit si výhody a nevýhody nezávislosti testování.	K2			X											
Kapitola 2	**Testování v průběhu životního cyklu vývoje software**															
2.1	**Testování v kontextu životního cyklu vývoje software**															
2.1.1	Vysvětlit vliv zvoleného životního cyklu vývoje softwaru na testování.	K2						X								
2.1.2	Vybavit si na osvědčené postupy při testování aplikovatelné na všechny typy SDLC.	K1						X								
2.1.3	Vybavit si příklady vývoje iniciovaného testy (test-first).	K1					X									
2.1.4	Shrnout, jaký vliv na testování může mít DevOps.	K2					X	X			X	X				

2.1.5	Vysvětlit přístup k testování shift-left.	K2					X	X								
2.1.6	Vysvětlit, jak je možné využít retrospektivy jako mechanismu pro zlepšování procesů.	K2					X					X				
2.2	**Úrovně testování a typy testů**															
2.2.1	Rozlišit mezi různými úrovněmi testování.	K2		X	X											
2.2.2	Rozlišit mezi různými typy testů.	K2		X												
2.2.3	Odlišit konfirmační testování od regresního testování.	K2		X												
2.3	**Testování údržby**															
2.3.1	Shrnout testování údržby a jeho spouštěče.	K2		X					X							
Kapitola 3	**Statické testování**															
3.1	**Základy statického testování**															
3.1.1	Určit typy produktů, které mohou být otestovány s použitím různých technik statického testování.	K1				X	X									
3.1.2	Vysvětlit hodnotu testování bílé skříňky.	K2	X			X	X									
3.1.3	Porovnat a uvést odlišnosti statického a dynamického testování.	K2				X	X									
3.2	**Proces zpětné vazby a revise**															
3.2.1	Identifikovat výhody včasné a časté zpětné	K1	X			X						X				

	vazby od zainteresovaných stran.															
3.2.2	Shrnout činnosti procesu revize.	K2			X	X										
3.2.3	Vybavit si, které odpovědnosti při provádění revizí jsou přiřazeny jednotlivým rolím.	K1				X								X		
3.2.4	Porovnat a uvést odlišnosti různých typů revizí.	K2		X												
3.2.5	Vybavit si faktory, které přispívají k úspěšné revizi.	K1					X							X		
Kapitola 4	**Testovací analýza a návrh testů**															
4.1	**Přehled technik testování**															
4.1.1	Rozlišit mezi technikami testování černé skříňky, bílé skříňky a technikami založenými na zkušenostech.	K2		X												
4.2	**Techniky testování černé skříňky**															
4.2.1	Použít techniku rozdělení tříd ekvivalence k odvození testovacích případů.	K3					X									
4.2.2	Použít techniku analýzy hraničních hodnot k odvození testovacích případů.	K3					X									
4.2.3	Použít techniku testování dle rozhodovací tabulky k odvození testovacích případů.	K3					X									

4.2.4	Použít techniku testování přechodů stavů k odvození testovacích případů.	K3					X									
4.3	**Techniky testování bílé skříňky**															
4.3.1	Vysvětlit techniku testování příkazů.	K2		X												
4.3.2	Vysvětlit techniku testování větví.	K2		X												
4.3.3	Vysvětlit přínos testování bílé skříňky.	K2	X	X												
4.4	**Techniky testování založené na zkušenostech**															
4.4.1	Vysvětlit techniku odhadování chyb.	K2		X												
4.4.2	Vysvětlit techniku průzkumného testování.	K2		X												
4.4.3	Vysvětlit techniku testování založenou na kontrolních seznamech.	K2		X												
4.5	**Přístupy k testování založené na spolupráci**															
4.5.1	Vysvětlit, jak psát uživatelské scénáře ve spolupráci s vývojáři a zástupci byznysu.	K2				X						X				
4.5.2	Kategorizovat různé možnosti psaní akceptačních kritérií.	K2										X				
4.5.3	Použít techniku vývoje řízeného akceptačními testy k odvození testovacích případů.	K3					X									
Kapitola 5	**Management testování**															
5.1	**Plánování testování**															

5.1.1	Ilustrovat na příkladech účel a obsah plánu testování.	K2		X					X							
5.1.2	Určit, jakou přidanou hodnotu má přítomnost testerů pro plánování iterací a plánování vydání.	K1	X									X		X		
5.1.3	Porovnat a uvést odlišnosti vstupních a výstupních kritérií.	K2				X		X								X
5.1.4	Použít techniky odhadování pro výpočet pracnosti potřebné při testování.	K3							X		X					
5.1.5	Použít prioritizaci testovacích případů.	K3							X		X					
5.1.6	Vybavit si koncept testovací pyramidy.	K1		X												
5.1.7	Shrnout testovací kvadranty a uvědomit si jejich vztah k úrovním testování a typům testů.	K2		X							X					
5.2	**Management rizik**															
5.2.1	Identifikovat úroveň rizika pomocí pravděpodobnosti a dopadu rizika.	K1							X						X	
5.2.2	Odlišit mezi projektovým a produktovým rizikem.	K2		X											X	
5.2.3	Vysvětlit, jak může analýza produktových rizik ovlivnit hloubku a rozsah testování.	K2					X				X				X	
5.2.4	Vysvětlit, jaká opatření lze přijmout v reakci na analyzovaná produktová rizika.	K2		X			X								X	

5.3	**Monitoring, řízení a dokončení testování**															
5.3.1	Vybavit si metriky využívané při testování.	K1									X					X
5.3.2	Shrnout účel a obsah reportů z testování a uvědomit si, pro koho jsou určeny.	K2					X				X					X
5.3.3	Ilustrovat na příkladech komunikaci stavu testování.	K2												X		X
5.4	**Konfigurační management**															
5.4.1	Shrnout, jak konfigurační management přispívá k testování.	K2					X		X							
5.5	**Management defektů**															
5.5.1	Připravit report o defektu.	K3		X						X						
Kapitola 6	**Testovací nástroje**															
6.1	**Nástroje pro podporu testování**															
6.1.1	Vysvětlit, jak různé typy testovacích nástrojů podporují testování.	K2					X									
6.2	**Výhody a rizika automatizace testů**															
6.2.1	Vybavit si výhody a rizika automatizace testů.	K1					X						X			

10 Příloha C – Poznámky k vydání

Učební osnovy ISTQB® základní úrovně (ISTQB® Foundation Syllabus) v4.0 jsou významnou aktualizací, která vychází z předchozí verze 3.1.1 osnov základní úrovně a z učebních osnov Agile Tester 2014. Z tohoto důvodu neexistují detailní poznámky k vydání (release notes) na úrovni jednotlivých kapitol a sekcí a přehled zásadních změn je uveden v této kapitole.

Kromě toho poskytuje ISTQB® (v samostatných poznámkách k vydání) trasovatelnost mezi studijními cíli (LO) učebních osnov základní úrovně ve verzi 3.1.1, studijními cíli učebních osnov Agile Tester verze 2014 a studijními cíli učebních osnov základní úrovně ve verzi 4.0, kde je navíc zdůrazněno, které studijní cíle byly přidány, aktualizovány nebo odstraněny.

V době, kdy byly tyto učební osnovy vytvořeny (2022–2023) bylo evidováno více než 1 000 000 osob ve více než 100 zemích, kteří absolvovali zkoušku základní úrovně. Celosvětově pak existuje více než 800 000 certifikovaných testerů. Za předpokladu, že si všichni z nich přečetli učební osnovy základní úrovně s cílem složit zkoušku, je možné říct, že se jedná pravděpodobně o nejčtenější dokument v oblasti testování vůbec. Tato aktualizace je vytvořena tak, aby tyto hodnoty respektovala a také, aby přispěla ke zlepšení názorů na úroveň kvality, kterou ISTQB® přináší globální testovací komunitě.

V této verzi byly všechny studijní cíle upraveny tak, aby byly atomické, a aby byla vytvořena 1:1 trasovatelnost mezi studijními cíli a kapitolami učebních osnov. Jinými slovy, učební osnovy neobsahují kapitoly (obsah), ke kterým neexistuje studijní cíl. Cílem je usnadnit čtení, porozumění, učení a překlad s důrazem na zvýšení praktické užitečnosti a rovnováhy mezi znalostmi a dovednostmi.

V této verzi došlo k následujícím změnám:

- Zmenšení celkové velikosti osnov. Učební osnovy nejsou učebnice, ale dokument, který slouží k nastínění základní kostry jakéhokoliv úvodního kurzu o testování softwaru včetně informací o tom, jaká témata by měla být pokryta a na jaké úrovni. To znamená následující:
  - Ve většině případů jsou z textu vyloučeny příklady. Úkolem poskytovatele školení je pak tyto příklady (včetně praktických cvičení) poskytnout během školení.
  - Byl dodržen princip „Velikost textu vzhledem ke K-úrovni“, který definuje maximální velikost textu pro studijní cíle na každé úrovni K (K1 = max. 10 řádků, K2 = max. 15 řádků, K3 = max. 25 řádků).
- Snížení počtu studijních cílů ve srovnání s osnovami základní úrovně (CTFL) verze 3.1.1 a Agile Tester (CTFL-AT) verze 2014:
  - 14 K1 cílů (FL v3.1.1 měla 15 cílů a AT 2014 6 cílů),
  - 42 K2 cílů (FL v3.1.1 měla 40 cílů a AT 2014 13 cílů),
  - 8 K3 cílů (FL v3.1.1 měla 7 cílů a AT 2014 8 cílů).
- K dispozici jsou rozsáhlejší odkazy na klasické a/nebo uznávané knihy a články o testování softwaru a souvisejících tématech.
- Hlavní změny v kapitole 1 (Základy testování)
  - Byla rozšířena a vylepšena podkapitola o dovednostech testerů.
  - Byla přidána podkapitola o týmovém přístupu (K1).
  - Podkapitola o nezávislosti testování byla přesunuta z kapitoly 5 do kapitoly 1.
- Hlavní změny v kapitole 2 (Testování v rámci SDLC)
  - Podkapitoly 2.1.1 a 2.1.2 byly přepracovány a vylepšeny, zároveň byly upraveny odpovídající studijní cíle.
  - Je kladen vyšší důraz na techniky vývoje iniciovaného testy (K1), přístupu shift-left (K2) a retrospektiv (K2).
  - Přidána nová podkapitola o testování v kontextu DevOps (K2).
  - Úroveň integračního testování byla rozdělena do dvou samostatných testovacích úrovní (testování integrace komponent a testování integrace systému).
- Hlavní změny v kapitole 3 (Statické testování)

  - Podkapitola o technikách revize spolu s odpovídajícím studijním cílem „(K3) Použít techniku revize“ byla odstraněna.
- Hlavní změny v kapitole 4 (Testovací analýza a návrh testů)
  - Testování případů užití bylo odstraněno (ale je stále předmětem osnov Advanced Test Analyst).
  - Vyšší důraz je kladen na techniky testování založených na spolupráci (přidán nový K3 studijní cíl o použití ATDD k odvození testovacích případů a dva nové K2 cíle o uživatelských scénářích a akceptačních kritériích).
  - Testování a pokrytí rozhodnutí nahrazeno testováním a pokrytím větví (za prvé, termín pokrytí větví se v praxi používá více; za druhé, různé standardy definují termín rozhodnutí odlišně od větve; za třetí, řeší to drobný, ale závažný nedostatek ze starých osnov základní úrovně verze 2018, kde je napsáno, že „100% pokrytí rozhodnutí znamená 100% pokrytí příkazů“ – tato věta není pravdivá v případě kódu bez podmínek nebo cyklů, tedy bez bodů rozhodnutí).
  - Byla vylepšena podkapitola o hodnotě testování bílé skříňky.
- Hlavní změny v kapitole 5 (Management testovacích aktivit)
  - Oddíl o testovacích strategiích/přístupech byl odstraněn.
  - Přidán nový K3 studijní cíl o technikách odhadování pracnosti pro testování.
  - Je kladen vyšší důraz na všeobecně známé agilní koncepty a nástroje při managementu testování: plánování iterací a vydání (K1), testovací pyramida (K1) a testovací kvadranty (K2).
  - Oddíl o managementu rizik je lépe strukturovaný popisem čtyř hlavních činností: identifikace rizik, posouzení rizik, zmírnění rizik a monitoring rizik.
- Hlavní změny v kapitole 6 (Testovací nástroje)
  - Rozsah textu o automatizací testů byl snížen z důvodů příliš vysoké odbornosti nevhodné pro učební osnovy základní úrovně.
  - Podkapitola o výběru nástrojů, provádění pilotních projektů a zavádění nástrojů do organizace byla odstraněna.