Učební osnovy – Certifikovaný tester základní úrovně

Verze 2018 v3.1

International Software Testing Qualifications Board

Upozornění o ochraně autorských práv

Historie změn

Verze	Datum	Poznámka
ISTQB CTFL 2018 CZ 3.1.1	22.3.2020	Opravy překlepů a drobných formálních chyb.
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 RC2	29.1.2019	Release candidate 2.
ISTQB CTFL 2018 CZ RC1	26.1.2019	Release candidate 1.
ISTQB CTFL 2018 CZ Beta	9.1.2019	Beta verze pro členy a partnery CaSTB.
ISTQB CTFL 2018 CZ Alfa	20.12.2018	Alfa verze pro členy CaSTB.
ISTQB CTFL 2011 CZ 1.0.0	15.1.2017	Oficiální vydání 1.0.0.
ISTQB CTFL 2011 CZ RC1	31.12.2016	Aktualizované vydání. Release candidate 1.
ISTQB CTFL 2011 Beta	15.6.2013	Aktualizované vydání. Beta verze.
ISTQB CTFL 2011 Beta 1	14.8.2008	Verze Beta 1 – nepublikovaná verze.

Historie změn (anglický originál)

Verze	Datum	Poznámka
ISTQB® 2018 V3.1	11-November-2019	Certified Tester Foundation Level Syllabus Maintenance Release with minor updates – see Release Notes
ISTQB® 2018	27-April-2018	Candidate general release version
ISTQB® 2011	1-Apr-2011	Certified Tester Foundation Level Syllabus Maintenance Release – see Release Notes
ISTQB® 2010	30-Mar-2010	Certified Tester Foundation Level Syllabus Maintenance Release – see Release Notes
ISTQB® 2007	01-May-2007	Certified Tester Foundation Level Syllabus Maintenance Release
ISTQB® 2005	01-July-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-Feb-1999	ISEB Software Testing Foundation Syllabus V2.0

Obsah

Poděkování (ISTQB®)

Tento dokument byl formálně vydán Valným shromážděním ISTQB® (11. listopadu 2019). Byl sepsán týmem ISTQB ve složení: Klaus Olsen (předseda), Meile Posthuma a Stephanie Ulrich.

Tým děkuje lokálním výborům za jejich připomínky týkající se Osnov základní úrovně 2018.

Osnovy základní úrovně (The Foundation Syllabus) 2018 byly sepsán týmem ISTQB® ve složení: Klaus Olsen (předseda), Tauhida Parveen (místopředsedkyně), Rex Black (projektový manažer), Debra Friedenberg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrichová, Marie Walsh a Eshraka Zakaria.

Tým děkuje týmu ve složení Rex Black a Dorothy Graham za jejich technickou úpravu, revizním týmům a lokálním výborům za jejich návrhy a vstupy. Následující osoby se zúčastnily revize a hlasování o těchto učebních osnovách: Tom Adams, Tobias Ahlgren, Xu Aiguo, Chris Van Bael, Katalin Balla, Graham Bath, Gualtiero Bazzana, Arne Becher, Veronica Belcher, Lars Hilmar Bjørstrup, Ralf Bongard, Armin Born, Robert Bornelind, Mette Bruhn-Pedersen, Geza Bujdoso, Earl Burba, Filipe Carlos, Young Jae Choi, Greg Collina, Alessandro Collino, Cui Zhe, Taz Daughtrey, Matthias Daigl, Wim Decoutere, Frans Dijkman, Klaudia Dussa-Zieger, Yonit Elbaz, Ofer Feldman, Mark Fewster, Florian Fieber, David Frei, Debra Friedenberg, Conrad Fujimoto, Pooja Gautam, Thorsten Geiselhart, Chen Geng, Christian Alexander Graf, Dorothy Graham, Michel Grandjean, Richard Green, Attila Gyuri, Jon Hagar, Kobi Halperin, Matthias Hamburg, Zsolt Hargitai, Satoshi Hasegawa, Berit Hatten, Wang Hongwei, Tamás Horváth, Leanne Howard, Chinthaka Indikadahena, J. Jayapradeep, Kari Kakkonen, Gábor Kapros, Beata Karpinska, Karl Kemminger, Kwanho Kim, Seonjoon Kim, Cecilia Kjellman, Johan Klintin, Corne Kruger, Gerard Kruijff, Peter Kunit, Hyeyong Kwon, Bruno Legeard, Thomas Letzkus, Alon Linetzki, Balder Lingegård, Tilo Linz, Hongbiao Liu, Claire Lohr, Ine Lutterman, Marek Majernik, Rik Marselis, Romanos Matthaios, Judy McKay, Fergus McLachlan, Dénes Medzihradszky, Stefan Merkel, Armin Metzger, Don Mills, Gary Mogyorodi, Ninna Morin, Ingvar Nordström, Adam Novak, Avi Ofer, Magnus C Ohlsson, Joel Oliviera, Monika Stocklein Olsen, Kenji Onishi, Francisca Cano Ortiz, Gitte Ottosen, Tuula Pääkkönen, Ana Paiva, Tal Pe'er, Helmut Pichler, Michaël Pilaeten, Horst Pohlmann, Andrew Pollner, Meile Posthuma, Vitalijs Puiso, Salvatore Reale, Stuart Reid, Ralf Reissing, Shark Ren, Miroslav Renda, Randy Rice, Adam Roman, Jan Sabak, Hans Schaefer, Ina Schieferdecker, Franz Schiller, Jianxiong Shen, Klaus Skafte, Mike Smith, Cristina Sobrero, Marco Sogliani, Murian Song, Emilio Soresi, Helder Sousa, Michael Sowers, Michael Stahl, Lucjan Stapp, Li Suyuan, Toby Thompson, Steve Toms, Sagi Traybel, Sabine Uhde, Stephanie Ulrich, Philippos Vakalakis, Erik van Veenendaal, Marianne Vesterdal, Ernst von Düring, Salinda Wickramasinghe, Marie Walsh, Søren Wassard, Hans Weiberg, Paul Weymouth, Hyungjin Yoon, John Young, Surong Yuan, Ester Zabar a Karolina Zmitrowicz.

Tým „International Software Testing Qualifications Board 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. Klíčový tým děkuje týmu revidujících a všem lokálním výborům za návrhy k současným učebním osnovám.

Tým „International Software Testing Qualifications Board Working Group Foundation Level“, verze 2011: Thomas Müller (předseda), Debra Friedenberg. Klíčový 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 lokálním výborům za návrhy k současným učebním osnovám.

Tým „International Software Testing Qualifications Board Working Group Foundation Level“, verze 2010: Thomas Müller (předseda), Rahul Verma, Martin Klonk a Armin Beer. Klíčový 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 lokálním výborům za jejich připomínky.

Tým „International Software Testing Qualifications Board Working Group Foundation Level“, verze 2007: Thomas Müller (předseda), Dorothy Graham, Debra Friedenberg a Erik van Veendendaal. Klíčový tým děkuje týmu revidujících (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson a Wonil Kwon) a všem lokálním výborům za návrhy.

Tým „International Software Testing Qualifications Board 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. Klíčový tým děkuje týmu revidujících a všem lokálním výborům za návrhy.

Poděkování (CaSTB)

Překlad do českého jazyka, verze 2018 v3.1: David Janota, Petr Neugebauer, Lukáš Piška 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 z roku 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. Mezinárodní výbor pro kvalifikaci testování softwaru (International Software Testing Qualifications Board, dále již jen ISTQB®) poskytuje tyto učební osnovy takto:

1. Členským výborům pro překlad do jejich lokálního 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 lokálním jazyce uzpůsobených studijním cílům těchto osnov.
3. Poskytovatelům školení pro přípravu výukových kurzů a určení vhodných výukových metod.
4. Uchazečům o certifikaci za účelem přípravy na certifikační zkoušku, a to buď jako součást výukového kurzu nebo samostatně.
5. Mezinárodní komunitě softwarového a systémového inženýrství za účelem podpory profese testování softwaru a systémů, rovněž jako zdroj pro odborné knihy a články.

ISTQB® může umožnit použití těchto osnov dalším subjektům i k jiným účelům, pokud si to tyto subjekty vyžádají a získají od ISTQB® předchozí písemný souhlas.

0.2 Certifikovaný tester základní úrovně pro testování softwaru

Základní úroveň kvalifikace je určena každému, kdo je zapojen do testování softwaru. To mohou být lidé v rolích testerů, testovacích analytiků, specialistů v oblasti testování, konzultantů, manažerů testování, akceptačních testerů a softwarových vývojářů. 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. Vlastnictví certifikátu základní úrovně opravňuje jeho držitele mimo jiné k zisku vyšších úrovní certifikace v oblasti testování softwaru.

Přehled učebních osnov ISTQB® základní úrovně ve verzi 2018 (ISTQB® Foundation Level Overview 2018) je samostatný dokument, jenž je v platnosti i pro tuto verzi Osnov základní úrovně (Foundation Level 2018 v3.1), a který poskytuje následující informace:

- profesní cíle učebních osnov,
- matice trasovatelnosti mezi profesními cíli a studijními cíli,
- shrnutí těchto učebních osnov.

0.3 Přezkoumatelné studijní cíle a kognitivní úrovně znalostí

Studijní cíle sledují profesní cíle a jsou použity k vytváření certifikačních zkoušek základní úrovně.

Obecně lze říct, že veškerý obsah těchto osnov je přezkoumatelný na úrovni K1 (viz dále), s výjimkou úvodu a příloh. To znamená, že kandidát může být požádán, aby rozpoznal, pamatoval si nebo si vzpomněl na klíčové slovo nebo pojem z jakékoliv 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.

Další podrobnosti a příklady studijních cílů jsou uvedeny v Příloze B.

Na začátku každé kapitoly (těsně pod názvem kapitoly) je uveden seznam klíčových slov. Všechny uvedené klíčové pojmy je nutné si zapamatovat (K1), a to i v případě, že nejsou výslovně uvedeny ve studijních cílech.

0.4 Certifikační zkouška základní úrovně

Zkouška pro získání certifikátu základní úrovně bude založena na těchto učebních osnovách. Odpovědi na zkouškové otázky mohou vyžadovat znalosti těchto osnov, a to z jedné nebo i více kapitol. Všechny části osnov mohou být součástí certifikační zkoušky (kromě úvodu a příloh). Standardy, knihy a jiné osnovy ISTQB® jsou uvedeny jako odkazy. Jejich obsah není součástí zkoušky s výjimkou toho, co je z těchto standardů, knih a jiných osnov ISTQB® přímo zahrnuto v těchto osnovách.

Formát zkoušky je test s více možnostmi odpovědí (multiple-choice). Test sestává z celkem 40 otázek. Pro úspěšné složení zkoušky je potřebné odpovědět správně alespoň na 65 % otázek (tj. alespoň 26 otázek).

Zkoušky je možné absolvovat v rámci akreditovaného vzdělávacího kurzu nebo nezávisle (např. v testovacím centru nebo na veřejné zkoušce). Absolvování akreditovaného vzdělávacího kurzu není podmínkou pro vykonání zkoušky.

0.5 Akreditace

Členský výbor ISTQB® může akreditovat poskytovatele školení, jehož studijní materiály se řídí těmito učebními osnovami. Pokyny pro akreditaci by měli poskytovatelé školení získat od daného členského výboru nebo orgánu, který akreditaci provádí. Akreditovaný kurz je následně uznán za vyhovující těmto osnovám. Do takového kurzu je možné zahrnout i ISTQB® zkoušku.

0.6 Úroveň detailu

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

- obecných instruktážních cílů popisujících záměr osnov,
- seznamu pojmů, které si studenti musí být schopni vybavit (vzpomenout si),
- studijních cílů pro každou oblast znalostí popisujících výsledek kognitivního učení, kterého má být dosaženo,
- popisu klíčových slov včetně odkazů na zdroje, jako jsou například použitá literatura a normy.

Obsah osnov není popisem 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 testování a technik, které lze použít pro všechny softwarové projekty, a to včetně agilních. Tyto učební osnovy neobsahují žádné specifické studijní cíle týkající se konkrétního životního cyklu vývoje software nebo metodiky, nicméně iniciují diskusi, jak tyto koncepty uplatnit v agilních projektech nebo jiných typech iterativně-inkrementálních či sekvenčních modelech životních cyklu vývoje softwaru.

0.7 Uspořádání učebních osnov

Součástí učebních osnov je celkem šest kapitol s přezkoumatelným obsahem. Nejvyšší úrovně nadpisu pro každou kapitolu určují celkovou dobu výuky pro danou kapitolu; časování není uvedeno pro nižší úrovně kapitoly. U akreditovaných vzdělávacích kurzů vyžadují tyto osnovy výuku v rozsahu nejméně 16,75 hodin rozdělených do šesti kapitol a to takto:

- Kapitola 1: 175 minut Základy testování.
- Kapitola 2: 100 minut Testování v průběhu životního cyklu vývoje softwaru.
- Kapitola 3: 135 minut Statické testování.
- Kapitola 4: 330 minut Techniky testování.
- Kapitola 5: 225 minut Management testování.
- Kapitola 6: 40 minut Nástroje pro podporu testování.

1 Základy testování — 175 minut

Klíčová slova

Pokrytí, ladění, defekt, omyl, selhání, kvalita, zajištění kvality, kořenová příčina, testovací analýza, testovací báze, testovací případ, dokončení testu, testovací podmínka, řízení testování, testovací data, návrh testů, provedení testů, implementace testování, monitoring testování, testovaný objekt, cíl testování, testovací orákulum, plánování testování, testovací procedura, proces testování, testovací sada, testování, testware, trasovatelnost, validace, ověřování.

Studijní cíle

1.1 Co je to testování?

FL-1.1.1 (K1) Identifikovat typické cíle testování.
FL-1.1.2 (K2) Rozlišovat mezi testováním a laděním (debugging).

1.2 Proč je testování nezbytné?

FL-1.2.1 (K2) Uvést příklady toho, proč je testování nezbytné.
FL-1.2.2 (K2) Popsat vztah mezi testováním a zajištěním kvality, a uvést příklady, jak testování přispívá k vyšší kvalitě.
FL-1.2.3 (K2) Rozlišovat mezi chybou, defektem a selháním.
FL-1.2.4 (K2) Rozlišovat mezi kořenovou příčinou defektu a jejími důsledky.

1.3 Sedm principů testování

FL-1.3.1 (K2) Vysvětlit sedm principů testování.

1.4 Proces testování

FL-1.4.1 (K2) Vysvětlit, jaký vliv má kontext na proces testování.
FL-1.4.2 (K2) Popsat testovací činnosti a příslušné úkoly v rámci procesu testování.
FL-1.4.3 (K2) Rozlišovat ty pracovní produkty, které podporují proces testování.
FL-1.4.4 (K2) Vysvětlit hodnotu udržování trasovatelnosti mezi testovací bází a testovacími pracovními produkty.

1.5 Psychologie testování

FL-1.5.1 (K1) Identifikovat psychologické faktory, které mohou ovlivnit úspěch testování.
FL-1.5.2 (K2) Vysvětlit rozdíl mezi podstatou uvažování, které je potřebné pro činnosti testování, a které je potřebné pro činnosti vývoje.

1.1 Co je to testování?

Softwarové systémy jsou nedílnou součástí života, a to od obchodních aplikací (např. v bankovnictví) až po spotřební zboží (např. automobily). Zkušenosti se softwarem, který nefungoval podle očekávání, má většina lidí. Software, který nefunguje správně, může způsobit mnoho problémů, včetně ztráty peněz, času nebo obchodní pověsti, dokonce může způsobit zranění nebo smrt. Testování softwaru je způsob, jakým lze posoudit kvalitu softwaru, zároveň pomáhá snížit rizika selhání softwaru v provozu.

Zažitým mýtem je, že testování je pouze provádění testů, tedy spouštění (testovaného) softwaru a následná kontrola výsledků. Jak je popsáno v kapitole 1.4, testování softwaru je proces, který zahrnuje mnoho různých aktivit, přičemž provádění testů (včetně kontroly výsledků) je pouze jednou z těchto činností. Proces testování zahrnuje také plánování testování, analýzu, návrh a implementaci testování, reportování průběhu testů a jejich výsledků a hodnocení kvality testovaného objektu.

Pokud testování vyžaduje i spuštění testované komponenty nebo systému, označujeme takové testování jako dynamické testování. V opačném případě, pokud testování testované komponenty nebo systému nevyžaduje její spuštění, nazýváme takové testování jako statické testování. Testování tak zahrnuje rovněž revize pracovních produktů, jakými jsou požadavky, uživatelské scénáře a zdrojový kód.

Dalším zažitým mýtem o testování je, že se zaměřuje výhradně na ověření požadavků, uživatelských scénářů nebo jiných specifikací. Testování zahrnuje kontrolu toho, zda systém odpovídá stanoveným požadavkům, ale také ověření, zda systém splňuje potřeby uživatelů a dalších zainteresovaných stran v jeho provozním prostředí.

Testovací aktivity jsou organizovány a prováděny rozdílně v různých životních cyklech vývoje softwaru (viz kapitola 2.1).

1.1.1 Typické cíle testování

Mezi cíle testování v každém projektu patří:

- Předcházet vzniku defektů pomocí ohodnocení pracovních produktů, jako jsou požadavky, uživatelské scénáře, návrh a kód.
- Ověřit, zda byly splněny všechny specifikované požadavky.
- Zkontrolovat, že je testovaný objekt kompletní a validovat, že funguje tak, jak uživatelé a zainteresované strany očekávají.
- Vytvořit důvěru v danou úroveň kvality testovaného objektu.
- Odhalit defekty a selhání, a tím snížit úroveň rizika nízké kvality softwaru.
- Poskytnout informace zúčastněným stranám v dostatečné míře tak, aby mohly činit kvalifikovaná rozhodnutí, zejména pokud jde o úroveň kvality testovaného objektu.
- Dodržet smluvní, právní nebo regulatorní požadavky nebo normy a/nebo ověřit, zda testovaný objekt dosahuje shody s takovými požadavky nebo normami.

Cíle testování se mohou lišit v závislosti na kontextu testované komponenty nebo systému, úrovni testování a modelu životního cyklu vývoje softwaru, například:

- Během testování komponent může být jedním z cílů odhalení co možná největšího počtu selhání tak, aby došlo k identifikaci příslušných defektů a ty mohly být včas opraveny. Dalším cílem může být zvýšení pokrytí kódu pomocí testů komponent.

- Během akceptačního testování může být jedním z cílů potvrzení, že systém funguje podle očekávání a splňuje požadavky. Dalším cílem akceptačního testování může být poskytnutí informací zúčastněným stranám týkající se rizika vydání v daném časovém okamžiku.

1.1.2 Testování a ladění (debugging)

Testování a ladění jsou rozdílné činnosti. Prováděním testů je možné odhalit selhání, která jsou způsobena defekty v softwaru. Ladění je vývojářská činnost, při které se tyto defekty hledají, analyzují a opravují. Následné konfirmační testování kontroluje, zda došlo v důsledku těchto oprav k jejich vyřešení. V některých případech jsou testeři zodpovědní za počáteční testování a závěrečné konfirmační testování, zatímco vývojáři jsou zodpovědni za ladění a integrační testování příslušných komponent (v rámci procesu průběžné integrace). V agilním vývoji, příp. i v jiných životních cyklech vývoje softwaru, mohou být i testeři zapojeni do ladění a testování komponent.

Pro další informace o konceptech softwarového testování viz norma *ČSN ISO/IEC/IEEE 29119-1*.

1.2 Proč je testování nezbytné?

Pečlivé testování komponent a systémů včetně jejich související dokumentace může přispět ke snížení rizika selhání během provozu. Zjištěné a následně opravené defekty zvyšují kvalitu komponent nebo systémů. Kromě toho může být testování softwaru vyžadováno také z důvodu splnění smluvních nebo zákonných požadavků či norem pro dané odvětví.

1.2.1 Jak testování přispívá k úspěchu

V průběhu historie výpočetní techniky bylo celkem běžné, že software a systémy, které byly nasazeny do provozu, obsahovaly defekty, které následně způsobily selhání nebo byly v rozporu s potřebami zúčastněných stran. Použití vhodných technik testování může snížit četnost defektů v takových problémových systémech, pokud jsou tyto techniky aplikovány s odpovídající úrovní znalostí testování. To znamená použití ve vhodných úrovních testování a na příslušných místech v životním cyklu vývoje softwaru, například:

- Zapojení testerů do revize požadavků nebo do zpřesňování uživatelských scénářů v těchto pracovních produktech může odhalit defekty. Identifikace a odstranění defektů v požadavcích snižuje riziko vývoje nesprávných nebo netestovatelných (užitných) vlastností.
- Úzká spolupráce testerů se systémovými návrháři během návrhu systému může umožnit oběma stranám tento návrh lépe pochopit, a to včetně odpovídajícího způsobu testování. To může snížit riziko výskytu prapůvodních defektů v návrhu a podpořit identifikaci potřebných testů již v počáteční fázi.
- Úzká spolupráce testerů s vývojáři už během vývoje může zvýšit vzájemné pochopení kódu a způsobu, jak jej otestovat. Toto zvýšené pochopení může snížit riziko výskytu defektů v kódu a v testech.
- Testeři ověřující a validující software před jeho vydáním mohou odhalit selhání, která by se jinak nezachytila, a podporují tím proces odstraňování defektů způsobující tato selhání (tj. ladění). Tím se zvyšuje pravděpodobnost, že software splní potřeby zainteresovaných stran a odpovídá požadavkům.

Kromě těchto příkladů přispívá dosažení definovaných cílů testování (viz kapitola 1.1.1) také k celkovému úspěchu vývoje a údržby softwaru.

1.2.2 Zajištění kvality a testování

Zatímco lidé často používají výraz *zajištění kvality* (nebo jen *QA*) pro testování, nejsou oba výrazy totožné, přestože spolu souvisí. Oba pojmy jsou součástí disciplíny nazývané management kvality. Management kvality zahrnuje všechny činnosti, které vedou a řídí organizaci s ohledem na kvalitu. Kromě jiného je součástí managementu kvality zajištění kvality a řízení kvality (quality control). Zajištění kvality je obvykle zaměřeno na dodržování správných procesů s cílem dosažení důvěry v to, že bude dosaženo odpovídající úrovně kvality. Pokud jsou procesy prováděny správně, pracovní produkty vytvořené těmito procesy jsou obecně kvalitnější, což přispívá k prevenci defektů. Kromě toho je používání analýzy kořenové příčiny pro odhalení a odstranění příčin defektů spolu s řádným používáním poznatků z retrospektiv pro zlepšování procesů důležitým faktorem pro efektivní zajištění kvality.

Řízení kvality zahrnuje různé činnosti, včetně testovacích aktivit, které podporují dosažení náležité úrovně kvality. Testovací aktivity jsou součástí celkového procesu vývoje nebo údržby softwaru. Zajištění kvality se týká správného provádění celého procesu, tudíž podporuje i správné testování. Jak je popsáno v kapitolách 1.1.1 a 1.2.1, testování přispívá k dosažení kvality několika způsoby.

1.2.3 Chyby, defekty a selhání

Člověk může udělat chybu a (omylem) zapříčinit vznik defektu (vady nebo bugu) v softwarovém kódu nebo v nějakém jiném souvisejícím pracovním produktu. Chyba, která způsobí zanesení defektu do jednoho pracovního produktu, může způsobit jinou chybu, která má za následek zanesení jiného defektu do souvisejícího pracovního produktu. Například chyba při definování požadavků může vést k defektu v nich obsaženému, což vede k chybě při programování, která způsobí defekt v kódu.

Pokud dojde ke spuštění defektu v kódu, může tento defekt způsobit (ne vždy a ne nutně) selhání. Aby například některé defekty způsobily selhání, musí existovat velmi specifické vstupy nebo vstupní podmínky. K takovým selháním pak může docházet zřídka nebo nikdy.

Chyby mohou nastat z mnoha důvodů, například:

- časová tíseň,
- lidská omylnost,
- nezkušení nebo nedostatečně kvalifikovaní členové projektu,
- nesprávná komunikace mezi členy projektu, včetně nesprávné komunikace o požadavcích a návrhu,
- složitost kódu, návrhu, architektury, prapůvodního řešeného problému a/nebo použité technologie,
- nepochopení vnitřních nebo vnějších systémových rozhraní, zejména pokud je jich velké množství,
- nové, neznámé technologie.

Kromě selhání způsobených defekty v kódu mohou být selhání způsobena také podmínkami prostředí. Příkladem může být radiace, elektromagnetické pole a znečištění, které mohou způsobit defekty ve firmwaru nebo ovlivnit spuštění softwaru změnou hardwarových podmínek.

Ne všechny neočekávané výsledky testu jsou nutně selháními. Někdy může docházet k falešně-pozitivním výsledkům (false-positives) kvůli chybám ve způsobu, jakým byly testy provedeny nebo kvůli defektům v testovacích datech, testovacím prostředí, jiném testwaru nebo ze zcela jiných důvodů. Může nastat také opačná situace, kdy podobné chyby nebo defekty povedou k falešně-negativním výsledkům. K falešně-negativním výsledkům dochází v případě, že testy neodhalí defekty,

které by odhalit měly; falešně-pozitivní výsledky jsou hlášeny jako defekty, které ve skutečnosti defekty nejsou.

1.2.4 Defekty, kořenové příčiny a následky

Kořenové příčiny defektů jsou prvotní akce nebo podmínky, které přispěly ke vzniku defektů. Identifikované defekty je možné analyzovat s cílem zjištění kořenové příčiny. Díky tomu lze zajistit, že se v budoucnu výskyt podobných defektů sníží. K zaměření na nejvýznamnější kořenové příčiny může přispět analýza kořenových příčin, jejíž cílem je zlepšení procesů; tím lze zabránit vzniku výrazného počtu budoucích defektů.

Předpokládejme například, že nesprávné platby úroků (z důvodu jediného chybného řádku v kódu) mají za následek stížnosti zákazníků. Chybný kód byl napsán pro uživatelský scénář, který byl nejednoznačný kvůli tomu, že vlastník produktu nepochopil, jak vypočítat úrok. Pokud existuje ve výpočtech úroků velké procento defektů a tyto defekty mají svou kořenovou příčinu v podobných nepochopeních, měli by být konkrétní vlastníci produktů vyškoleni v oblasti výpočtů úroků s cílem tyto defekty do budoucna redukovat.

V tomto příkladu stížnosti zákazníků představují následky, nesprávné platby úroků představují selhání. Chybný výpočet v kódu je defekt, který vyplynul z původního defektu, kterým je nejednoznačnost
v daném uživatelském scénáři. Kořenovou příčinou původního defektu byla nedostatečná znalost ze strany vlastníka produktu, což vedlo k tomu, že tento vlastník produktu udělal chybu při psaní uživatelského scénáře. Proces analýzy kořenových příčin je popsaný v učebních osnovách *ISTQB-CTEL-TM* a *ISTQB-CTEL-ITP*.

1.3 Sedm principů testování

Během posledních 50 let byla formulována řada principů, které poskytují obecný návod společný pro všechny druhy testování.

1. Testování ukazuje přítomnost defektů, nikoli jejich nepřítomnost

Testování může ukázat, že defekty jsou přítomny, ale nemůže prokázat, že žádné defekty neexistují. Testování snižuje pravděpodobnost, že v softwaru zůstaly neodhalené defekty, nicméně pokud nejsou žádné defekty nalezeny, není tím prokázána jeho správnost.

2. Kompletní testování není možné

Testování všeho (všech kombinací vstupů a vstupních podmínek) není možné s výjimkou triviálních případů. Místo snahy o kompletní testování je lepší se zaměřit na analýzu rizik, vhodné techniky testování a prioritizaci.

3. Včasné testování šetří čas a peníze

Pro včasné zjištění defektů by měly být statické i dynamické testovací aktivity zahájeny co nejdříve v životním cyklu vývoje softwaru. Včasné testování je někdy označováno jako *shift left* (posun doleva). Testování v rané fázi vývoje softwaru pomáhá snížit nebo eliminovat budoucí nákladné změny (viz kapitola 3.1).

4. Shlukování defektů

Většinu defektů zjištěných během testování před vydáním obsahuje obvykle malé množství modulů nebo jsou defekty v těchto modulech zodpovědné za většinu provozních poruch. Předpokládané a skutečně pozorované shluky defektů během testování nebo v provozu jsou významným přínosem pro analýzu rizik, která se využívá k vhodnému zaměření testovacích aktivit (jak zmiňuje Princip 2).

5. Vyvarování se pesticidnímu paradoxu

Pokud se stejné testy opakují neustále dokola, neodhalí nakonec žádné nové defekty. Pro odhalení nových defektů může být nezbytné provést změny ve stávajících testech a testovacích datech, příp. je třeba vytvořit testy nové (testy již nejsou efektivní pro odhalování defektů, stejně jako pesticidy již po čase neúčinkují při ničení hmyzu). V některých případech (jako je například automatizované regresní testování) má však pesticidní paradox pozitivní přínos, kterým je relativně malý počet regresních defektů.

6. Testování je závislé na kontextu

Testování se provádí odlišně v různých kontextech. Například řídicí průmyslový bezpečnostně kritický software se testuje jinak než mobilní aplikace pro e-commerce. Dalším příkladem je testování v agilním projektu, kde testování probíhá odlišně od testování na projektu s využitím sekvenčního životního cyklu vývoje softwaru (viz kapitola 2.1).

7. Nepřítomnost chyb je klam

Některé organizace očekávají, že testeři dokážou provést všechny možné testy a najít všechny možné defekty, ale Principy 1 a 2 nám říkají, že to možné není. Dalším omylem (tj. mylnou představou) je očekávání, že pouhým nalezením a odstraněním velkého počtu defektů lze zajistit úspěch systému. Například i přes důkladné testování všech specifikovaných požadavků a odstranění všech zjištěných defektů by mohl přesto vzniknout obtížně použitelný systém, který by nesplňoval potřeby a očekávání uživatelů, příp. by nepřinesl takovou (vyšší) hodnotu v porovnání s jinými konkurenčními systémy.

Příklady těchto a dalších testovacích principů viz *Myers 2011*, *Kaner 2002, Weinberg 2008 a Beizer 1990*.

1.4 Proces testování

Neexistuje žádný univerzální proces testování softwaru, existují však běžné sady testovacích činností, bez nichž bude méně pravděpodobné, že testování dosáhne stanovených cílů. Tyto sady testovacích činností tvoří proces testování. Vhodný a konkrétní proces testování softwaru v dané situaci závisí na mnoha faktorech. Jaké činnosti testování jsou zahrnuty do tohoto procesu testování, jak jsou tyto činnosti prováděny a kdy se tyto činnosti vyskytují, to vše může být předmětem diskusí v rámci strategie testování v dané organizaci.

1.4.1 Proces testování v kontextu

Faktory, které ovlivňují proces testování v dané organizaci, zahrnují následující položky (nejsou však omezeny pouze na ně):

- model životního cyklu vývoje softwaru a použitá metodika projektového řízení,
- zvažované úrovně a typy testů,
- projektová a produktová rizika,
- sféra byznysu,
- provozní omezení, včetně (avšak nejenom):
    - rozpočtu a zdrojů,
    - časových rozvržení,
    - složitosti,

  - smluvních a regulatorních požadavků,
- politiky a postupy organizace,
- shoda s požadovanými interními a externími normami.

Následující kapitoly popisují všeobecné aspekty organizačních procesů testování z hlediska těchto bodů:

- testovací činnosti a úkoly,
- pracovní produkty z testování,
- trasovatelnost mezi testovací bází a pracovními produkty z testování.

Je velmi užitečné, pokud má testovací báze (pro jakoukoli zvažovanou úroveň nebo typ testování) definovaná a měřitelná kritéria pokrytí. Kritéria pokrytí mohou fungovat efektivně jako klíčové ukazatele výkonnosti (KPI – key performance indicators) pro řízení činností, které prokazují dosažení cílů softwarového testování (viz kapitola 1.1.1).

Například pro mobilní aplikaci může testovací báze obsahovat seznam požadavků a seznam podporovaných mobilních zařízení. Každý požadavek je součástí testovací báze, každé podporované zařízení je součástí testovací báze. Kritéria pokrytí mohou vyžadovat existenci nejméně jednoho testovacího případu pro každou položku testovací báze. Jakmile jsou tyto testy provedeny, jsou zúčastněné strany informovány o tom, zda jsou splněny specifikované požadavky, a zda byla na podporovaných zařízeních zjištěna selhání.

Pro další informace o procesech testování viz norma *ČSN ISO/IEC/IEEE 29119-2*.

1.4.2 Testovací činnosti a úkoly

Proces testování se skládá z následujících hlavních skupin činností:

- plánování testování,
- monitoring a řízení testování,
- testovací analýza,
- návrh testů,
- implementace testování,
- provedení testů,
- dokončení testování.

Každá hlavní skupina se pak skládá ze základních činností, které budou popsány v níže uvedených podkapitolách. Každá dílčí činnost obsahuje několik individuálních úkolů, které se mohou lišit v závislosti na konkrétním projektu nebo vydání.

Dále platí, že i když se mnohé z těchto hlavních skupin činností mohou jevit jako logicky seřazené za sebou, často jsou prováděny iterativně. Agilní vývoj například zastřešuje malé iterace sestávající z návrhu softwaru, vývoje a testování, přičemž vše je vykonáváno nepřetržitě, a to za podpory průběžného plánování. Proto i testovací činnosti probíhají v rámci tohoto přístupu k vývoji softwaru iterativně a průběžně. Dokonce i v sekvenčním modelu vývoje softwaru může postupná logická posloupnost hlavních skupin činností obsahovat překryvy, kombinace, souběžnosti, případně může dojít k vynechání některé položky. Z toho důvodu je obvykle vyžadováno přizpůsobit tyto hlavní skupiny činností kontextu systému a projektu.

Plánování testování

Plánování testování zahrnuje činnosti, které definují cíle testování a přístup k testování pro splnění testovacích cílů v rámci omezení daných kontextem (např. stanovení vhodných technik testování a úkolů a/nebo sestavení harmonogramu testů s ohledem na dodržení termínu). Plány testování mohou být přepracovány na základě zpětné vazby z monitorovacích a řídicích činností. Plánování testování je dále vysvětleno v kapitole 5.2.

Monitoring a řízení testování

Monitoring testování se zaměřuje na průběžné porovnání skutečného a plánovaného postupu prací pomocí sledovaných metrik, které jsou definovány v plánu testování. Řízení testování zahrnuje zavedení opatření nezbytných k dosažení cílů určených plánem testování (tyto cíle mohou být v průběhu času aktualizovány). Monitoring a řízení testování je podporováno vyhodnocením výstupních kritérií, které jsou v některých modelech životního cyklu vývoje softwaru označovány jako definice hotového (Definiton of Done – DoD), viz učební osnovy *ISTQB-CTFL-AT*. Mezi vyhodnocení výstupních kritérií pro provedení testů v rámci dané úrovně testování patří:

- kontrola výsledků testů a záznamů z testování na základě stanovených kritérií pokrytí,
- posouzení úrovně kvality komponent nebo systému na základě výsledků testů a záznamů z testování,
- určení, zda je zapotřebí dalších testů (např. pokud testy původně určené pro dosažení určité úrovně pokrytí produktových rizik nebyly dostačující, což by vyžadovalo vytvoření a provedení dodatečných testů).

Postup prací při testování oproti plánu se komunikuje zainteresovaným stranám v reportech o postupu prací v testování, včetně odchylek od plánu a informací pro případné rozhodnutí o zastavení testování.

Monitoring a řízení testování jsou dále vysvětleny v kapitole 5.3.

Testovací analýza

Během testovací analýzy se analyzuje testovací báze za účelem identifikace testovatelných užitných vlastností (features) a zároveň se určí odpovídající testovací podmínky. Jinými slovy, testovací analýza určuje „co se má testovat“ ve smyslu měřitelných kritérií pokrytí.

Mezi hlavní činnosti testovací analýzy patří:

- analýza testovací báze, která je vhodná pro odpovídající úroveň testování jako například:
    - specifikace požadavků – byznysové, funkcionální nebo systémové požadavky, uživatelské scénáře, eposy (epics), případy užití nebo další podobné pracovní produkty, které specifikují požadované funkcionální nebo nefunkcionální chování komponenty nebo systému,
    - informace o návrhu a implementaci – dokumenty nebo diagramy architektury systému nebo softwaru, návrhové specifikace, grafy toků volání, diagramy modelování (např. UML diagramy nebo diagramy vztahů a entit), specifikace rozhraní nebo další podobné pracovní produkty, které specifikují strukturu komponenty nebo systému,
    - implementace samotné komponenty nebo systému včetně kódu, dotazů a metadat databáze nebo rozhraní,

    - reporty o analýze rizik, které mohou brát v úvahu funkcionální, nefunkcionální a strukturální aspekty komponenty nebo systému,
- hodnocení testovací báze a položek testování s cílem identifikace různých typů defektů, například:
    - nejednoznačnosti,
    - opomenutí,
    - nekonzistence,
    - nepřesnosti,
    - rozpory,
    - nadbytečnosti,
- identifikace vlastností a sad vlastností (feature set) určených pro testování,
- definice a stanovení priorit testovacích podmínek pro každou vlastnost na základě analýzy testovací báze a zohlednění funkcionálních, nefunkcionálních a strukturálních charakteristik, dalších byznysových a technických faktorů a úrovně rizik,
- zachycení obousměrné trasovatelnosti mezi každou položkou testovací báze a souvisejícími testovacími podmínkami (viz kapitoly 1.4.3 a 1.4.4).

V procesu testovací analýzy může být užitečné použití technik testování černé skříňky, bílé skříňky a technik založených na zkušenostech (viz kapitola 4) s cílem definování správných a přesných důležitých testovacích podmínek spolu se snížením pravděpodobnosti jejich opomenutí.

V některých případech jsou produktem testovací analýzy testovací podmínky, které mohou být použity jako cíle testování v testovacích listinách (test charters). Testovací listiny jsou typickými pracovními produkty používanými v některých typech testování založeném na zkušenostech (viz kapitola 4.4.2). V případě, že jsou tyto cíle testování trasovatelné směrem k testovací bázi, je možné měřit pokrytí dosaženého během tohoto typu testování.

Identifikace defektů v průběhu testovací analýzy je významným potenciálním benefitem zejména v případě, kdy se nepoužívá žádný jiný proces revize a/nebo pokud je proces testování s procesem revize úzce spjatý. Takové aktivity v rámci testovací analýzy nejenže ověřují, zda jsou požadavky konzistentní, správně vyjádřené a úplné, ale zároveň validují, zda požadavky správně zachycují potřeby zákazníků, uživatelů a dalších zainteresovaných stran. Existují také techniky jako vývoj řízený chováním (BDD – behavior driven development) a vývoj řízený akceptačními testy (ATDD – acceptance tests driven development), které zahrnují vytváření testovacích podmínek a testovacích případů z uživatelských scénářů a akceptačních kritérií ještě před kódováním. Tyto techniky rovněž ověřují, validují a detekují defekty v těchto uživatelských scénářích a akceptačních kritériích (viz učební osnovy *ISTQB-CTFL-AT*).

Návrh testů

V rámci návrhu testů jsou testovací podmínky zpracovány ve formě obecných (high-level) testovacích případů, jejich sad a dalšího testwaru. Zatímco testovací analýza dává odpověď na otázku „co se má testovat?“, návrh testů dává odpověď na otázku „jak testovat?“

Mezi hlavní činnosti návrhu testů patří:

- návrh a stanovení priorit testovacích případů a jejich sad,
- identifikace potřebných podpůrných testovacích dat pro testovací podmínky a testovací případy,

- návrh testovacího prostředí a identifikace potřebné infrastruktury a nástrojů,
- zachycení obousměrné trasovatelnosti mezi testovací bází, testovacími podmínkami a testovacími případy (viz kapitola 1.4.4).

Zpracování testovacích podmínek do podoby testovacích případů a jejich sad v rámci návrhu testů často zahrnuje použití různých technik testování (viz kapitola 4).

Stejně jako u testovací analýzy, může také návrh testů vést k identifikaci podobných typů defektů v testovací bázi a stejně jako u testovací analýzy, identifikace defektů při návrhu testů je důležitým potenciálním benefitem.

Implementace testování

Během implementace testování je vytvořen (a/nebo dokončen) testware potřebný k provedení testů včetně sestavení posloupnosti provádění testovacích případů do testovacích procedur. Zatímco návrh testů dává odpověď na otázku „jak testovat?“, implementace testování dává odpověď na otázku „máme nyní k dispozici vše potřebné pro provedení testů?“

Mezi hlavní činnosti implementace testování patří:

- definice a prioritizace testovacích procedur a případně vytvoření automatizovaných testovacích skriptů,
- vytvoření testovacích sad z testovacích procedur a případně automatizovaných testovacích skriptů,
- zařazení testovacích sad v rámci harmonogramu provádění testů takovým způsobem, aby bylo dosaženo efektivního provedení testů (viz kapitola 5.2.4),
- vytvoření testovacího prostředí, a to i případně včetně sad testovacího vybavení (tzv. test harnesses), virtualizace služeb, simulátorů a dalších položek infrastruktury a ověření, že vše potřebné bylo řádně nastaveno,
- příprava testovacích dat a zajištění jejich správné integrace do testovacího prostředí,
- ověření a aktualizace obousměrné trasovatelnosti mezi testovací bází, testovacími podmínkami, testovacími případy, testovacími procedurami a testovacími sadami (viz kapitola 1.4.4). Jednotlivé úkoly se při návrhu testů a implementaci testování často kombinují.

U průzkumného testování a obecně u jiných typů testování založeného na zkušenostech může docházet k tomu, že návrh testů, jejich implementace, a i jejich dokumentování je součástí provádění testů. Průzkumné testování může být založeno na testovacích listinách (vytvořených během testovací analýzy) a provádění průzkumných testů běží současně s jeho návrhem a implementací (viz kapitola 4.4.2).

Provedení testů

Během provedení testů se testovací sady provádějí podle definovaného harmonogramu provádění testů.

Mezi hlavní činnosti provedení testů patří:

- zaznamenávání identifikátorů (IDs) a verzí jednotlivých položek testování nebo testovaných objektů, testovacích nástrojů a testwaru,
- provádění testů buď ručně nebo pomocí nástrojů pro provedení testů,
- porovnávání skutečných a očekávaných výsledků,

- analýza anomálií pro určení jejich pravděpodobných příčin (např. k selháním může docházet v důsledku defektů v kódu, ale zároveň může docházet k falešně-pozitivním výsledkům, viz kapitola 1.2.3),
- hlášení defektů na základě pozorovaných selhání (viz kapitola 5.6),
- zaznamenávání výsledků provedení testů (např. test prošel, test selhal, test je blokovaný),
- opakování testovacích činností, a to buď jako důsledek akce vykonané v souvislosti s nějakou anomálií nebo jako součást plánovaného testování (např. provedení opraveného testu, konfirmační testování a/nebo regresní testování),
- ověření a aktualizace obousměrné trasovatelnosti mezi testovací bází, testovacími podmínkami, testovacími případy, testovacími procedurami a výsledky testů.

Dokončení testování

Dokončení testování zahrnuje činnosti, které shromažďují data z ukončených testovacích činností s cílem konsolidace zkušeností, testwaru a dalších důležitých informací. K dokončení testování dochází v rámci projektových milníků, například když je vydán softwarový systém, dokončí se (nebo zruší) testovací projekt, ukončí se iterace v agilním projektu, dokončí se úroveň testování nebo dojde k dokončení servisního vydání (maintenance release).

Mezi hlavní činnosti dokončení testování patří:

- kontrola, zda jsou všechny reporty o defektech uzavřeny, zadání změnových požadavků nebo položek produktového backlogu vztažených k zadání defektů, které nebudou řešeny,
- vytvoření souhrnného reportu z testování, který by měl být komunikován zainteresovaným stranám,
- dokončení a archivace testovacího prostředí, testovacích dat, testovací infrastruktury a dalšího testwaru pro pozdější opakované použití,
- předání testwaru týmům, které mají na starost údržbu, případně i dalším projektovým týmům a/nebo jiným zainteresovaným stranám, které by ho mohly dále využít,
- analýza ponaučení získaných z dokončených testovacích činností s cílem stanovit potřebné změny pro budoucí iterace, vydání a projekty,
- použití získaných informací s cílem zlepšit zralost procesu testování.

1.4.3 Pracovní produkty z testování

Pracovní produkty z testování (test work products) jsou vytvářeny v rámci procesu testování. Stejně jako existují významné rozdíly ve způsobu, jakým organizace implementují proces testování, existují také výrazné rozdíly v typech pracovních produktů vytvořených v průběhu tohoto procesu, způsobech, jakými jsou tyto pracovní produkty organizovány a spravovány, a rovněž i v jejich názvosloví. Tyto učební osnovy se řídí výše uvedeným procesem testování a pracovními produkty popsanými v těchto osnovách a definovaných zároveň ve Slovníku pojmů ISTQB® (*ISTQB® Glossary*). Více informací lze nalézt v normě *ČSN ISO/IEC/IEEE 29119-3*.

Mnoho z pracovních produktů z testování popsaných v této části osnov lze získat a řídit pomocí nástrojů pro management testování a nástrojů pro řízení defektů (viz kapitola 6).

Pracovní produkty příslušné k plánování testování

Pracovní produkty související s plánováním testování obvykle obsahují jeden nebo více plánů testování. Plán testování obsahuje informace o testovací bázi, se kterou jsou zároveň prostřednictvím trasovatelnosti provázány další pracovní produkty (viz níže a dále kapitola 1.4.4),

jakožto i výstupní kritéria (nebo definice hotového), která budou použita během monitoringu a řízení testování. Plány testování jsou dále popsány v kapitole 5.2.

Pracovní produkty příslušné k monitoringu a řízení testování

Pracovní produkty související s monitoringem a řízením testování obvykle zahrnují různé typy reportů z testování, včetně reportů o postupu prací v testování (vytvářených průběžně a/nebo v pravidelných intervalech) a souhrnných reportů z testování (vytvářených po dosažení určitého milníku). Veškeré reporty z testování by měly obsahovat podrobnosti relevantní danému příjemci. Zároveň by měly poskytovat informace o postupu prací v testování k danému datu, včetně shrnutí výsledků prováděných testů a to hned, jakmile jsou k dispozici.

Pracovní produkty související s monitoringem a řízením testování by se měly zabývat i otázkami managementu projektů jako je dokončování úkolů, přidělování a využívání zdrojů a pracnost.

Monitoring a řízení testování včetně pracovních produktů příslušných k těmto aktivitám jsou dále vysvětleny v kapitole 5.3 těchto učebních osnov.

Pracovní produkty příslušné k testovací analýze

Pracovní produkty související s testovací analýzou zahrnují definované testovací podmínky se stanovenými prioritami. Každá z testovacích podmínek je ideálně obousměrně trasovatelná ke specifické položce testovací báze, kterou pokrývá. U průzkumného testování může testovací analýza zahrnovat vytvoření testovacích listin. Testovací analýza může také přispět k odhalení a reportování defektů v testovací bázi.

Pracovní produkty příslušné k návrhu testů

Výstupem aktivit návrhu testů jsou testovací případy a sady testovacích případů používaných při provádění testovacích podmínek definovaných v rámci testovací analýzy. Často se doporučuje navrhnout obecné testovací případy bez konkrétních hodnot pro vstupní data a bez očekávaných výsledků. Tyto obecné testovací případy jsou pak opakovaně použitelné ve vícero cyklech testování s různými konkrétními daty, přičemž stále vhodně dokumentují rozsah testovacího případu. V ideálním případě je každý testovací případ obousměrně trasovatelný směrem k testovací podmínce (příp. testovacím podmínkám), kterou (příp. které) pokrývá.

Návrh testů také vede k:

- návrhu a/nebo identifikaci potřebných testovacích dat,
- návrhu testovacího prostředí,
- identifikaci infrastruktury a nástrojů.

Rozsah, v jakém jsou tyto výstupy zdokumentovány, se však může výrazně lišit.

Pracovní produkty příslušné k implementaci testování

Mezi pracovní produkty související s implementaci testování patří:

- testovací procedury a určení pořadí těchto testovacích procedur,
- testovací sady,
- harmonogram provádění testů.

V ideálním případě je možné po dokončení implementace testování prokázat dosažení kritérií pro pokrytí stanovených v plánu testování, a to prostřednictvím obousměrné trasovatelnosti mezi testovacími procedurami a konkrétními položkami testovací báze skrze testovací případy a testovací podmínky.

V některých případech zahrnuje implementace testování vytvoření pracovních produktů, které využívají nástroje (např. virtualizace služeb) nebo jsou využívány nástroji (např. automatizované testovací skripty).

Implementace testování může také vést k vytvoření a ověření testovacích dat a testovacího prostředí. Komplexnost dokumentace poskytující výsledky ověření dat a/nebo prostředí se může v různých situacích lišit.

Testovací data slouží pro přiřazení konkrétních hodnot ke vstupům a očekávaným výsledkům testovacích případů. Tyto konkrétní hodnoty spolu s výslovnými instrukcemi k jejich použití transformují obecné testovací případy (high-level) do proveditelných specifických testovacích případů (low-level). Stejný obecný testovací případ je možné provádět s různými testovacími daty na různých verzích vydání testovaného objektu. Konkrétní očekávané výsledky přidružené ke konkrétním testovacím datům jsou identifikovány pomocí testovacího orákula (test oracle).

Při průzkumném testování mohou být některé pracovní produkty související s návrhem testů a implementací testování vytvořeny až během provedení testů, nicméně rozsah, v jakém jsou průzkumné testy (a jejich trasovatelnost na konkrétní položky testovací báze) dokumentovány, se může výrazně lišit.

Testovací podmínky definované v rámci testovací analýzy mohou být během implementace testování dále zpřesňovány.

Pracovní produkty příslušné k provádění testů

Mezi pracovní produkty související s prováděním testů patří:

- dokumentace o stavu jednotlivých testovacích případů nebo testovacích procedur (např. připraven ke spuštění, prošel, selhal, byl blokován, byl záměrně přeskočen atd.),
- reporty o defektech (viz kapitola 5.6),
- dokumentace k položkám testování, testovaným objektům, testovacím nástrojům a testwaru použitých při testování.

V ideálním případě je možné v okamžiku ukončení provádění testů určit a reportovat stav každé položky testovací báze prostřednictvím obousměrné trasovatelnosti k přidružené testovací proceduře (příp. k testovacím procedurám). Například můžeme říct, pro které požadavky prošly všechny plánované testy, pro které požadavky testy selhaly a/nebo existují k nim přidružené defekty, a u kterých požadavků existují testy, které stále čekají na spuštění. Tímto lze ověřit, zda byla splněna kritéria pokrytí, a zároveň to umožňuje informovat o výsledcích testů způsobem, který je pro zúčastněné strany srozumitelný.

Pracovní produkty příslušné k dokončení testování

Pracovní produkty související s dokončením testování zahrnují souhrnné reporty z testování, opatření vedoucí ke zlepšení následných projektů nebo iterací, změnové požadavky nebo položky v produktovém backlogu nebo dokončený testware.

1.4.4 Trasovatelnost mezi testovací bází a pracovními produkty z testování

Jak již bylo uvedeno v kapitole 1.4.3, pracovní produkty z testování a označení těchto pracovních produktů se může významně lišit případ od případu. Bez ohledu na tyto odlišnosti je pro zavedení efektivního sledování a řízení testování důležité zajistit vytvoření a následnou údržbu trasovatelnosti mezi každou položkou testovací báze a různými pracovními produkty z testování vztaženými k této položce, a to v průběhu celého procesu testování (jak je popsáno výše). Mimo vyhodnocení pokrytí testů podporuje správná trasovatelnost zároveň i následující aspekty:

- analýzu dopadu změn,
- zajištění možné kontroly testování (audit),
- splnění kritérií ve vztahu ke správě IT,
- lepší srozumitelnost reportů o postupu prací v testování a souhrnných reportů z testování poskytujících informace o stavu položek testovací báze (např. požadavky, u kterých testy prošly, požadavky, u kterých testy selhaly, a požadavky, u kterých testy stále čekají na spuštění),
- prezentování technických aspektů testování zúčastněným stranám ve formě, které rozumí,
- poskytování informací pro posouzení kvality produktu, schopnosti procesu a postupu prací v projektu ve vztahu k byznysovým cílům.

Některé nástroje pro management testování poskytují modely pracovních produktů z testování, které odpovídají části nebo i všem pracovním produktům z testování popsaných v této kapitole. Některé organizace si vytvářejí vlastní systémy řízení pro správu pracovních produktů a zajištění trasovatelnosti na míru svým potřebám.

1.5 Psychologie testování

Vývoj softwaru (včetně jeho testování) se týká i lidských bytostí, z toho důvodu má lidská psychika významný vliv na testování softwaru.

1.5.1 Lidská psychika a testování

Odhalování defektů během statického testování (například revize požadavků nebo schůzka pro zpřesňování uživatelských scénářů) nebo odhalování selhání v rámci dynamického testování bývá často vnímáno jako kritika produktu a jeho autora. Jeden z faktorů psychologie osobnosti označovaný pojmem konfirmační zkreslení (nebo též potvrzovací zkreslení, angl. confirmation bias) popisuje tendenci člověka obtížně přijímat informace, které jsou v rozporu s jeho stávajícími názory. Například pokud vývojáři očekávají, že jejich kód je správný, může u nich docházet ke konfirmačnímu zkreslení, které ztěžuje přijetí faktu, že kód obsahuje defekty. Kromě konfirmačního zkreslení existují i jiné kognitivní zkreslení (cognitive biases) způsobující to, že lidé mohou obtížně pochopit nebo přijmout informace získané pomocí testování. Dalším obvyklým lidským znakem je to, že viní nositele špatných zpráv (a informace získané testováním často takové špatné zprávy obsahují).

V důsledku těchto psychologických faktorů mohou někteří lidé vnímat testování jako destruktivní aktivitu, přestože významně přispívá k zdárnému průběhu projektu a kvalitě produktu (viz kapitoly 1.1 a 1.2). Při snaze snížit tato vnímání by informace o defektech a selháních měly být komunikovány konstruktivně. Tímto způsobem lze snížit napětí mezi testery a analytiky, vlastníky produktů, návrháři a vývojáři. Následující uvedená doporučení platí pro statické i dynamické testování.

Testeři a manažeři testování musí mít dobré schopnosti v oblasti mezilidských vztahů, aby mohli efektivně sdělovat informace o defektech, selháních, výsledcích testů, postupu prací v testování a rizicích, a aby mohli také vytvářet pozitivní vztahy s kolegy. Mezi způsoby správné komunikace patří:

- Snažte se spolupracovat, ne bojovat. Připomeňte každému, že vašim společným cílem je lepší kvalita systémů.
- Zdůrazněte přínosy testování. Například autorům pracovních produktů mohou informace o defektech pomoci zlepšit nejen tyto produkty, ale také jejich dovednosti. Pro firmu zase platí, že defekty nalezené a opravené během testování mohou ušetřit čas a peníze a mohou také snížit celkové riziko (nižší) kvality produktu.

- Výsledky testů a další zjištění komunikujte neutrálně se zaměřením na fakta, aniž byste kritizovali osobu, která vadnou položku vytvořila. Pište objektivní a věcné reporty o defektech a zjištění ověřujte.
- Pokuste se pochopit, jak se druhá osoba cítí, a snažte se určit důvody, které mohou mít za následek negativní reakci na tyto informace.
- Ujistěte se, že druhá osoba pochopila, co bylo sděleno (a také se ujistěte, že jste porozuměli tomu, co druhá osoba říká).

Typické cíle testování byly popsány výše (viz kapitola 1.1). Jasně definovaná (a správná) sada cílů testování má důležité psychologické důsledky. Většina lidí má sklon sladit své plány a chování s cíli stanovenými týmem, managementem a dalšími zainteresovanými stranami. Je zároveň důležité, aby se testeři drželi těchto cílů s minimálními osobními předsudky.

1.5.2 Smýšlení (mindset) testera a vývojáře

Vývojáři a testeři obvykle přemýšlejí jinak. Hlavním cílem vývoje je navrhnout a vytvořit produkt. Jak bylo uvedeno výše, cíle testování zahrnují mimo jiné ověření a validaci produktu a nalezení defektů před vydáním. Jedná se o rozdílné cíle, které vyžadují odlišná smýšlení. Propojením těchto odlišných smýšlení dohromady lze dosáhnout vyšší úrovně kvality produktu.

Smýšlení odráží individuální předpoklady a preferované metody pro rozhodování a řešení problémů. Smýšlení testera by mělo zahrnovat zvídavost, profesionální pesimismus, kritický pohled, smysl pro detail a motivaci pro správnou a pozitivní komunikaci a vztahy. Nabytými zkušenostmi se smýšlení testera rozšiřuje a dozrává.

Smýšlení vývojáře může obsahovat některé prvky smýšlení testera, nicméně úspěšní vývojáři se často spíše zajímají o navrhování a tvorbu řešení, než že by uvažovali nad tím, co by mohlo být na těchto řešeních špatně. Navíc je pro ně z důvodu konfirmačního zkreslení obtížné uvědomit si chyby, kterých se sami dopustí.

Se správným smýšlením jsou vývojáři schopni otestovat svůj vlastní kód. Různé modely životního cyklu vývoje softwaru často mají i různé způsoby organizace testerů a testovacích činností. Pokud jsou některé testovací činnosti prováděny nezávislými testery, zvyšuje se efektivita detekce defektů, což je zvláště důležité u rozsáhlých, složitých nebo bezpečnostně kritických systémů. Nezávislí testeři přinášejí úhel pohledu, který je odlišný od pohledu autorů pracovních produktů (tj. byznysových analytiků, vlastníků produktů, návrhářů a vývojářů), jelikož mají odlišné kognitivní předsudky oproti autorům.

2 Testování v průběhu životního cyklu vývoje softwaru	**100 minut**

Klíčová slova

Akceptační testování, alfa testování, beta testování, testování související se změnami, krabicový software (COTS – commercial off-the-shelf), integrační testování komponent, testování komponent, konfirmační testování, smluvní akceptační testování, funkcionální testování, analýza dopadu, integrační testování, testování údržby, nefunkcionální testování, provozní akceptační testování, regresní testování, regulatorní akceptační testování, sekvenční vývojový model (životního cyklu), systémové integrační testování, systémové testování, testovací báze, testovací případ, testovací prostředí, úroveň testování, testovaný objekt, cíl testování, typ testu, uživatelské akceptační testování, testování bílé skříňky.

Studijní cíle

2.1 Modely životního cyklu vývoje softwaru

FL-2.1.1 (K2) Vysvětlit vztahy mezi vývojovými a testovacími aktivitami při vývoji softwaru v rámci jeho životního cyklu.

FL-2.1.2 (K1) Určit důvody, proč musí být modely životního cyklu vývoje softwaru přizpůsobeny kontextu projektových a produktových charakteristik.

2.2 Úrovně testování

FL-2.2.1 (K2) Porovnat různé úrovně testování z pohledu cílů, testovací báze, testovaných objektů, typických defektů a selhání, a přístupů a odpovědností.

2.3 Typy testů

FL-2.3.1 (K2) Porovnat funkcionální a nefunkcionální testování a testování bílé skříňky.

FL-2.3.2 (K1) Uvědomit si, že funkcionální a nefunkcionální testování a testování bílé skříňky probíhá na kterékoliv úrovni testování.

FL-2.3.3 (K2) Porovnat účel konfirmačního testování a regresního testování.

2.4 Údržbové testování

FL-2.4.1 (K2) Shrnout spouštěče údržbového testování.

FL-2.4.2 (K2) Popsat úlohu analýzy dopadu při údržbovém testování.

2.1 Modely životního cyklu vývoje softwaru

Model životního cyklu vývoje softwaru popisuje druhy činností prováděných v každé fázi projektu vývoje softwaru a jak tyto činnosti k sobě logicky a chronologicky patří. Existuje řada různých modelů životního cyklu vývoje softwaru a každý z nich vyžaduje jiný přístup k testování.

2.1.1 Vztah mezi vývojem a testováním softwaru

Důležitou součástí práce testera je seznámit se s obecnými modely životního cyklu vývoje softwaru s cílem volby vhodných testovacích aktivit.

V jakémkoliv modelu životního cyklu vývoje softwaru existuje několik charakteristik správného testování:

- Ke každé vývojové aktivitě existuje odpovídající testovací činnost.
- Každá úroveň testování má své specifické cíle.
- Analýza a návrh testů pro danou úroveň testování začínají v průběhu odpovídající vývojové aktivity.
- Testeři se účastní diskusí s cílem definice a zpřesňování požadavků a návrhu, také se zapojují do revize pracovních produktů (například požadavků, návrhu, uživatelských scénářů apod.) a to hned, jakmile jsou k dispozici první koncepty (drafts).

Bez ohledu na to, jaký model životního cyklu vývoje softwaru je použit, testovací činnosti by měly být zahájeny už v raných fázích životního cyklu dle principu včasného testování.

Tyto učební osnovy kategorizují obecné modely životního cyklu vývoje softwaru takto:

- sekvenční modely vývoje,
- iterativně-inkrementální modely vývoje.

Sekvenční model vývoje popisuje proces vývoje softwaru jako lineární, sekvenční tok činností. To znamená, že zahájení další fáze vývoje následuje vždy po dokončení fáze předchozí. Teoreticky se tedy fáze nepřekrývají, i když v praxi je výhodné získat zpětnou vazbu pro následující fázi dříve (tedy ještě před skončením fáze předchozí).

Ve vodopádovém sekvenčním modelu (waterfall) jsou vývojové aktivity (jako například analýza požadavků, návrh, kódování, testování) prováděny a dokončovány jedna po druhé. Podle tohoto modelu tedy k testování dochází až po dokončení všech předcházejících vývojových fází.

Oproti tomu V-model integruje proces testování v průběhu celého vývojového procesu na základě principu včasného testování. V-model navíc obsahuje takové úrovně testování, které vždy souvisí s příslušnou vývojovou fází, což také podporuje princip včasného testování (viz kapitola 2.2). V tomto modelu probíhá provádění testů v jednotlivých úrovních sekvenčně, ale v některých případech může také docházet k překrývání.

Sekvenční modely vývoje předpokládají existenci kompletní sady vlastností (features) na začátku vývoje, přičemž od tohoto okamžiku pak trvá měsíce až roky, než se výsledný software dostane k zainteresovaným stranám a uživatelům.

Inkrementální vývoj předpokládá definování požadavků, návrh, sestavování a testování systému po částech, což znamená, že se jednotlivé vlastnosti softwaru vyvíjejí postupně (přírůstkově neboli inkrementálně). Velikost těchto přírůstků či inkrementů se může lišit – některé metody pracují s většími přírůstky, zatímco jiné s menšími. Přírůstek může být i malá změna v uživatelském rozhraní nebo využití nového dotazu (do databáze).

K iterativnímu vývoji dochází, pokud jsou skupiny vlastností specifikovány, navrženy, vyvinuty a testovány společně v sériích cyklů obvykle pevně dané délky. Iterace mohou rovněž zahrnovat změny vlastností vyvinutých v předchozích iteracích společně se změnami v rozsahu projektu jako celku. V každé iteraci se dodává funkční software, který je rostoucí podmnožinou sady vlastností a tento růst pokračuje až do vytvoření konečné podoby softwaru nebo ukončení vývoje.

Mezi různé metodiky inkrementálního vývoje patří:

- Rational Unified Process (RUP): Iterace jsou relativně dlouhé (například dva až tři měsíce) a přírůstky jsou tedy analogicky také velké, například mohou představovat dvě nebo tři skupiny souvisejících vlastností.
- Scrum: Každá iterace je obvykle relativně krátká (například několik hodin, dní nebo týdnů) a přírůstky jsou tedy analogicky malé, například drobná vylepšení a/nebo dvě až tři nové vlastnosti.
- Kanban: Realizuje se iteracemi o pevně dané délce, případně úplně bez nich. Výsledkem může být dodání jednoho vylepšení nebo jedné užitné vlastnosti v rámci dodání, případně je možno nové vlastnosti seskupovat a vydat je pak současně.
- Spirálový model: Spočívá v tvorbě experimentálních přírůstků, z nichž některé mohou být následně zásadně přepracovávány nebo dokonce v následujících fázích vývoje zcela vypuštěny.

Při vývoji komponent nebo systémů pomocí těchto metod dochází často k překryvu a opakování jednotlivých úrovní testování během celého vývoje. V ideálním případě je každá vlastnost během svého vývoje testována v několika úrovních testování, a to až do okamžiku konečné dodávky. V některých případech týmy využívají přístupy průběžného dodávání (continuous delivery) nebo průběžného nasazování (continuous deployment). Tyto přístupy znamenají velkou míru automatizace v několika úrovních testování. Vývoj s využitím těchto metod zahrnuje rovněž koncept samoorganizujících se týmů, který umožňuje měnit způsob testování a také vztahy mezi testery a vývojáři.

Tyto metody vytváří postupně rostoucí systém, který může být dodáván koncovým uživatelům po dílčích vlastnostech, po iteracích, nebo tradičnějším způsobem vydáním celého produktu (major release). Bez ohledu na to, zda jsou jednotlivé přírůstky softwaru uvolněny koncovým uživatelům k užívání nebo ne, s postupně rostoucím systémem roste i význam regresního testování.

Na rozdíl od sekvenčních modelů mohou iterativně-inkrementální modely dodávat použitelný software už po prvních týdnech či dokonce dnech vývoje, ale kompletní produkt splňující veškeré požadavky se může vyvíjet měsíce nebo dokonce roky.

Pro další informace o testování softwaru v kontextu agilního vývoje viz učební osnovy *ISTQB-CTFL-AT*, *Black 2017*, *Crispin 2008* a *Gregory 2015*.

2.1.2 Modely životního cyklu vývoje softwaru v kontextu

Modely životního cyklu vývoje softwaru musí být voleny a přizpůsobovány kontextu projektu a produktovým charakteristikám. Vhodný model životního cyklu vývoje softwaru by měl být vybrán a přizpůsoben cílům projektu, typu vyvíjeného produktu, byznysovým prioritám (například doba zavádění produktu na trh – time-to-market) a identifikovaným produktovým a projektovým rizikům. Například vývoj a testování malého interního administrativního systému by se měl lišit od vývoje a testování bezpečnostně kritického systému jako je například systém řízení brzd v automobilovém průmyslu. Jiným příkladem může být nekvalitní komunikace mezi členy týmu z důvodů organizačních nebo kulturních problémů, která může brzdit iterativní vývoj.

V závislosti na kontextu projektu může být nutné kombinovat a přeorganizovat úrovně testování a/nebo testovací aktivity. Například při integraci komerčního krabicového softwaru (COTS) do většího systému může kupující testovat interoperabilitu (schopnost spolupráce) v úrovni systémového integračního testování (např. integrace do infrastruktury a jiných systémů) a v úrovni akceptačního testování (funkcionálního a/nebo nefunkcionálního a uživatelského či provozního akceptačního testování). Více informací lze nalézt v kapitole 2.2 (pojednávající o úrovních testování) a kapitole 2.3 (popisující typy testů).

Kromě toho lze kombinovat i samotné modely životního cyklu vývoje softwaru. Například V-model lze využít pro vývoj a testování back-end systémů a jejich integraci, zatímco agilní model vývoje lze využít pro vývoj a testování front-endových uživatelských rozhraní (UI) a funkcionalit. Prototypování je možné využívat v raných fázích projektu s uplatněním modelu inkrementálního vývoje po skončení experimentální fáze.

Systémy Internetu věcí (Internet of Things, IoT), které jsou tvořeny mnoha různými objekty (např. zařízení, produkty a služby) často uplatňují pro jednotlivé objekty různé modely životního cyklu vývoje softwaru, což představuje velkou výzvu pro správu vývojových verzí takových systémů. Kromě toho životní cyklus vývoje softwaru pro tyto objekty klade velký důraz na pozdější fáze životního cyklu po jeho zavedení do provozního užívání (jako jsou fáze provozu, aktualizace a vyřazování z provozu).

Důvody, proč musí být modely vývoje softwaru přizpůsobeny kontextu projektu a charakteristikám produktu, mohou být:

- Rozdílnost produktových rizik systémů (složitý versus jednoduchý projekt)
- Součástí projektu nebo programu může být více organizačních jednotek (důsledkem je kombinace sekvenčního a agilního vývoje)
- Krátká doba pro dodání produktu na trh (sloučení úrovní testování a/nebo zařazení různých typů testů do dílčích úrovní testování).

2.2 Úrovně testování

Úrovně testování jsou skupiny testovacích činností, které jsou organizovány a řízeny společně. Každá úroveň testování je instancí procesu testování tvořenou činnostmi popsanými v kapitole 1.4 a prováděnými ve vztahu k softwaru v dané úrovni vývoje, od jednotlivých jednotek nebo komponent až po celé systémy, nebo případně i systémy systémů. Úrovně testování souvisejí s dalšími činnosti v rámci životního cyklu vývoje softwaru. Úrovně testování používané v těchto učebních osnovách jsou:

- testování komponent,
- integrační testování,
- systémové testování,
- akceptační testování.

Úrovně testování jsou charakterizovány následujícími atributy:

- specifické cíle,
- testovací báze, od níž jsou odvozovány testovací případy,
- testovaný objekt (tj. to, co se testuje),

- typické defekty a selhání,
- specifické přístupy a odpovědnosti.

Pro každou úroveň testování je potřeba vhodné testovací prostředí. Pro akceptační testování je například vhodné testovací prostředí odpovídající produkčnímu prostředí, zatímco pro testování komponent vývojáři běžně používají vlastní vývojové prostředí.

2.2.1 Testování komponent

Cíle testování komponent

Testování komponent (často nazývané také jednotkové testování nebo testování modulů) se zaměřuje na komponenty, které lze testovat samostatně. Cíle testování komponent zahrnují:

- snížení rizik (např. rizika výskytu defektů),
- ověření, zda je funkcionální a nefunkcionální chování komponenty takové, jak bylo navrženo a specifikováno,
- budování důvěry v kvalitu komponenty,
- nalezení defektů v komponentě,
- prevence proniknutí defektu do vyšších úrovní testování.

V některých případech, zejména u inkrementálně-iterativních modelů vývoje (např. agilního), kde průběžně dochází ke změnám v kódu, hrají významnou roli automatizované regresní testy komponent. Tyto testy posilují důvěru v to, že provedené změny neměly negativní dopad na neměněné funkcionalitě.

Testování komponent často probíhá odděleně od zbytku systému v závislosti na modelu životního cyklu vývoje softwaru a na systému. Může vyžadovat existenci virtualizace služeb, sad testovacího vybavení (test harnesses), stubů a ovladačů (drivers). Testování komponent může zahrnovat testování funkcionalit (například správnosti výpočtů), nefunkcionálních charakteristik (například hledání úniků paměti) a strukturálních vlastností (například testování rozhodnutí).

Testovací báze

Příklady pracovních produktů, které lze použít jako testovací bázi pro testování komponent, zahrnují:

- podrobný návrh,
- kód,
- datový model,
- specifikace komponenty.

Testované objekty

Typické objekty pro testování komponent zahrnují:

- komponenty, jednotky nebo moduly,
- kód a datové struktury,
- třídy,
- databázové moduly.

Typické defekty a selhání

Příklady typických defektů a selhání pro testování komponent zahrnují:

- nesprávná funkcionalita (například jiná, než jaká je popsána ve specifikaci návrhu),
- problémy datových toků,
- nesprávný kód a logika.

Defekty jsou nejčastěji opravovány hned, jakmile jsou nalezeny, většinou bez formálního managementu defektů. Pokud ale vývojáři defekty reportují, poskytují důležitou informaci pro analýzu kořenové příčiny a zlepšení procesu.

Specifické přístupy a odpovědnosti

Testování komponent obvykle provádí vývojář, který vytvořil kód. Testování vždy vyžaduje přístup k testovanému kódu. Vývojáři mohou pracovat na vývoji komponent, ale mohou se také zabývat nacházením a opravováním defektů. Po implementaci kódu komponenty vývojáři často navrhují a provádějí i jeho testy. V agilním vývoji může dokonce implementace automatizovaných testovacích případů pro danou komponentu předcházet vlastní implementaci kódu. Typickým příkladem takového přístupu je vývoj řízený testy (test-driven development, TDD).

Vývoj řízený testy je vysoce iterativní přístup. Je založený na cyklech, ve kterých se vyvíjí automatizované testovací případy, následně se implementují a integrují malé části kódu, provádí se testy komponent a opravují se všechny problémy společně s přepracováváním kódu (refactoring). Tento proces pokračuje, dokud nejsou všechny komponenty dokončeny a všechny testy neproběhnou úspěšně. Vývoj řízený testy je příkladem přístupu někdy nazývaného testy nejdříve (test-first). I když vývoj řízený testy vznikl v rámci extrémního programování (eXtreme Programming, XP), rozšířil se i do jiných forem agilních i sekvenčních životních cyklů vývoje (viz učební osnovy *ISTQB-CTFL-AT*).

2.2.2 Integrační testování

Cíle integračního testování

Integrační testování je zaměřeno na interakce mezi komponentami nebo systémy. Cíle integračního testování zahrnují:

- snížení rizika,
- ověření, zda funkcionální a nefunkcionální chování rozhraní odpovídá tomu, jak bylo navrženo a specifikováno,
- budování důvěry v kvalitu rozhraní,
- nalezení defektů (které mohou být v samotném rozhraní, v komponentách nebo systémech),
- prevence proniknutí defektu do vyšších úrovní testování.

Podobně jako v případě testování komponent v některých případech automatizované regresní integrační testy poskytují důvěru, že změny nenarušily funkčnost stávajících rozhraní, komponent či systémů.

Tyto učební osnovy popisují dvě různé úrovně integračního testování, které mohou být realizovány na testovaných objektech různých velikostí:

- Integrační testování komponent je zaměřeno na interakce a rozhraní mezi integrovanými komponentami. Integrační testování komponent se provádí po testech komponent a je obvykle automatizované, v případě iterativně-inkrementálního vývoje jsou testy integrace komponent obvykle součástí procesu průběžné integrace.
- Systémové integrační testování je zaměřeno na interakce a rozhraní mezi systémy, balíčky a mikroslužbami. Systémové integrační testování může pokrývat i interakce s externími

organizacemi a rozhraními, které tyto externí organizace poskytují (například webovými službami). V takovém případě organizace zajišťující vývoj softwaru nemá vliv na externí rozhraní, což může způsobovat různé problémy při testování (například je potřeba zajistit, aby byly odstraněny defekty v kódu externí organizace, které blokující testování s cílem zajištění testovací prostředí). Systémové integrační testování může být prováděno po systémových testech nebo souběžně s nimi (jak v sekvenčním modelu, tak iterativně-inkrementálního modelu vývoje).

Testovací báze

Mezi příklady pracovních produktů, které lze použít jako testovací bázi pro integrační testování patří:

- návrh softwaru a systému,
- sekvenční diagramy,
- specifikace rozhraní a komunikačních protokolů,
- případy užití,
- architektura na úrovni komponent nebo systémů,
- pracovní toky,
- definice externích rozhraní.

Testované objekty

Mezi typické objekty pro integrační testování patří:

- subsystémy,
- databáze,
- infrastruktura,
- rozhraní,
- aplikační rozhraní (API),
- mikroslužby.

Typické defekty a selhání

Mezi příklady typických defektů a selhání pro integrační testování komponent patří:

- nesprávná data, chybějící data nebo nesprávné kódování dat,
- nesprávné sekvence nebo časování volání na rozhraní,
- nesoulad rozhraní,
- selhání komunikace mezi komponentami,
- neodchycené nebo nesprávně ošetřené selhání komunikace mezi komponentami,
- nesprávné předpoklady týkající se významu, jednotek nebo hraničních dat předávaných mezi komponentami.

Mezi příklady typických defektů a selhání pro systémové integrační testování patří:

- nejednotné struktury předávaných zpráv mezi systémy,
- nesprávná data, chybějící data nebo nesprávné kódování dat,
- nesoulad rozhraní,

- selhání komunikace mezi systémy,
- neodchycené nebo nesprávně ošetřené selhání komunikace mezi systémy,
- nesprávné předpoklady týkající se významu, jednotek nebo hraničních dat předávaných mezi systémy,
- nesplnění povinných předpisů o bezpečnosti.

Specifické přístupy a odpovědnosti

Integrační testování komponent a systémů by mělo být zaměřeno na vlastní integraci. Například při integraci modulu A s modulem B by testy měly být zaměřeny na komunikaci mezi moduly, nikoliv na funkcionality jednotlivých modulů, jelikož ty byly testovány v rámci testování komponent. Při integraci systému X se systémem Y by testy měly být zaměřeny na komunikaci mezi systémy, nikoliv na funkcionality jednotlivých systémů, jelikož ty byly testovány v rámci systémového testování. Použitelné jsou funkcionální, nefunkcionální a strukturální typy testů.

Integrační testování komponent je obvykle úkolem vývojářů. Systémové integrační testování je obecně úkolem testerů. Testeři provádějící systémové integrační testování by měli rozumět systémové architektuře a měli by se účastnit plánování integrace.

Pokud jsou integrační testy a strategie integrace plánovány předtím, než jsou vytvořeny komponenty nebo systémy, mohou být tyto komponenty nebo systémy vytvářeny v takovém pořadí, aby bylo testování co nejefektivnější. Systematické integrační strategie mohou být založené na architektuře systému (jako shora-dolů, zdola-nahoru), funkcionálních úlohách, sekvencích zpracování transakcí nebo na jiných aspektech systému nebo komponent. Pro jednodušší izolaci defektů a jejich včasnou detekci by měla být integrace inkrementální (po malých počtech komponent nebo systémů) spíše než hromadná v podobě „velkého třesku“ (tj. integrace všech komponent nebo systémů v jediném kroku). Analýza rizik nejsložitějších rozhraní může napomoci se správným zacílením integračního testování.

Čím je větší rozsah současně integrovaných komponent, tím je složitější izolovat defekty na specifickou komponentu nebo systém. Důsledkem může být zvýšení rizik a potřeba dodatečného času pro řešení problémů. To je jeden z důvodů, proč se průběžná integrace, kdy je software integrován po jednotlivých komponentách (tj. funkcionální integrace), stala běžnou praxí. Průběžná integrace často zahrnuje automatizované regresní testování, ideálně v několika úrovních testování.

2.2.3 Systémové testování

Cíle systémového testování

Systémové testování je zaměřeno na chování a schopnosti celého systému nebo produktu. Často zahrnuje end-to-end testování funkčností systému a testování nefunkcionální chování, které při jejich realizaci systém vykazuje. Mezi cíle systémového testování patří:

- snížení rizika,
- ověření, zda funkcionální a nefunkcionální chování systému odpovídá tomu, jak bylo navrženo a specifikováno,
- ověření, zda je systém kompletní a bude fungovat, jak se od něj očekává,
- budování důvěry v kvalitu systému jako celku,
- nalezení defektů,
- prevence proniknutí defektu do vyšších úrovní testování nebo do produkce.

Pro určité systémy může být také cílem testování ověření kvality dat. Podobně jako v případech testování komponent a integračního testování, automatizované regresní systémové testy podporují důvěru, že změny nenarušily integritu stávajících funkcionalit nebo schopností end-to-end činnosti systému. Systémové testování často poskytuje informace využívané zainteresovanými stranami při rozhodování o uvolnění systému do provozu. Systémové testování (resp. jeho detailní podoba) může být také vynuceno regulatorními požadavky a požadavky příslušných norem.

Testovací prostředí by mělo ideálně odpovídat cílovému nebo produkčnímu prostředí.

Testovací báze

Mezi příklady pracovních produktů, které lze použít jako testovací bázi pro systémové testování, patří:

- specifikace systémových a softwarových požadavků (funkcionálních a nefunkcionálních),
- reporty z analýzy rizik,
- případy užití,
- eposy a uživatelské scénáře,
- modely chování systému,
- stavové diagramy,
- systémové a uživatelské příručky.

Testované objekty

Mezi typické objekty pro systémové testování patří:

- aplikace,
- hardwarové/softwarové systémy,
- operační systémy,
- testovaný systém (SUT, system under test),
- konfigurace systému a konfigurační data.

Typické defekty a selhání

Mezi příklady typických defektů a selhání pro systémové testování patří:

- nesprávné výpočty,
- nesprávné nebo neočekávané funkcionální nebo nefunkcionální chování systému,
- nesprávné řízení dat nebo datové toky v systému,
- nesprávné nebo neúplné provádění end-to-end funkcionálních úloh,
- nesprávné fungování systému v jeho systémovém prostředí,
- systém nefunguje tak, jak je popsáno v systémových a uživatelských příručkách

Specifické přístupy a odpovědnosti

Systémové testování by mělo být zaměřeno na celkové chování systému jako celku, jak funkcionální, tak nefunkcionální. Systémové testování by mělo využívat techniky (viz kapitola 4) nejvhodnější pro testované aspekty systému. Například může být vytvořena rozhodovací tabulka pro ověření, zda funkcionální chování systému odpovídá popisu danému v byznysových pravidlech.

Systémové testování je obvykle prováděno nezávislými testery, kteří se ve velké míře spoléhají na specifikace. Defekty ve specifikacích (např. chybějící uživatelské scénáře, nesprávně definované byznysové požadavky apod.) mohou vést k nedorozuměním nebo sporům o očekávaném chování systému. Takové situace mohou vést k falešně-pozitivním (false-positive) nebo k falešně-negativním (false-negative) výsledkům testování, které zdržují a snižují účinnost detekce defektů. Včasné zapojení testerů do zpřesňování uživatelských scénářů a/nebo do aktivit statického testování (např. různé typy revizí) pomáhá snižovat výskyt takových situací.

2.2.4 Akceptační testování

Cíle akceptačního testování

Akceptační testování, podobně jako systémové testování, se obvykle zaměřuje na chování a schopnosti celého systému nebo produktu. Mezi cíle akceptačního testování patří:

- vytvoření důvěry v kvalitu systému jako celku,
- ověření, zda je systém kompletní a bude fungovat, jak se od něj očekává,
- ověření, zda funkcionální a nefunkcionální chování systému odpovídá tomu, jak bylo specifikováno.

Akceptační testování může poskytovat informace pro posouzení připravenosti systému k nasazení do provozního prostředí a zahájení používání zákazníkem (resp. koncovým uživatelem). Během akceptačního testování mohou být nalezeny defekty, nicméně nacházení defektů není jeho cílem. Nalezení velkého počtu defektů během akceptačního testování může být v některých případech považováno za významné projektové riziko. Akceptační testování (resp. jeho detailní podoba) může být také vynuceno regulatorními požadavky a požadavky příslušných norem.

Mezi obvyklé formy akceptačního testování patří:

- uživatelské akceptační testování,
- provozní akceptační testování,
- smluvní a regulatorní akceptační testování
- alfa a beta testování.

Následující sekce obsahují popis těchto forem akceptačního testování.

Uživatelské akceptační testování (UAT)

Uživatelské akceptační testování systému je běžně zaměřeno na ověřování vhodnosti systému k použití zamýšlenými uživateli v reálném nebo simulovaném provozním prostředí. Hlavním cílem je vybudovat důvěru uživatelů v to, že systém bude splňovat jejich potřeby a požadavky a umožní provádět podnikové procesy hladce s minimálními náklady a riziky.

Provozní akceptační testování (OAT)

Akceptační testování systému provozním personálem nebo správci systému se obvykle provádí v (simulovaném) produkčním prostředí. Testy jsou zaměřeny na provozní aspekty a mohou zahrnovat:

- testování zálohování a obnovy ze záloh,
- testování instalace, odinstalace a upgrade,
- testování obnovy po havárii,
- testování správy uživatelů,

- testování údržby systému,
- testování úloh načítání a migrace dat,
- kontroly zranitelnosti zabezpečení systému,
- testování výkonnosti.

Hlavním cílem provozního akceptačního testování je vytvořit důvěru v to, že provozovatelé a správci systému budou schopni zajistit správné chování a fungování systému pro uživatele v provozním prostředí, a to i za výjimečných nebo obtížných podmínek.

Smluvní a regulatorní akceptační testování

Smluvní akceptační testování je vykonáváno vůči smluvním akceptačním kritériím pro provoz softwaru vyvinutého na zakázku. Akceptační kritéria by měla být definována v době, kdy zúčastněné strany odsouhlasí kontrakt. Smluvní akceptační testování nejčastěji provádějí uživatelé nebo nezávislí testeři.

Regulatorní testování je vykonáváno vůči jakýmkoliv předpisům, které musí být dodrženy, jako například vládní, právní nebo bezpečnostní předpisy. Regulatorní akceptační testování nejčastěji provádějí uživatelé nebo nezávislí testeři, někdy za přítomnosti zástupců regulatorních orgánů.

Hlavním cílem smluvního a regulatorního akceptačního testování je vybudovat důvěru v dodržení smluvních a regulatorních požadavků.

Alfa testování a beta testování

Alfa a beta testování obvykle využívají vývojáři komerčního krabicového softwaru (COTS, commercial off-the-shelf), kteří chtějí získat zpětnou vazbu od potenciálních nebo stávajících uživatelů, zákazníků a provozovatelů před uvedením softwarového produktu na trh. Alfa testování se provádí na pracovišti vývojové organizace, nikoliv však vývojovým týmem, ale potenciálními nebo stávajícími zákazníky a provozovateli systému nebo nezávislým testovacím týmem. Beta testování provádějí potenciální nebo stávající zákazníci nebo provozovatelé systému na svých vlastních pracovištích. Beta testování může následovat po alfa testování nebo mu alfa testování ani nemusí předcházet.

Společným cílem alfa i beta testování je vybudování důvěry mezi potenciálními i stávajícími zákazníky a provozovateli, že mohou systém používat za normálních každodenních podmínek, a ve svém provozním prostředí k dosahování svých cílů hladce s minimálními náklady a riziky. Dalším cílem může být nalezení defektů v souvislosti s podmínkami a prostředím, kde bude systém používán (zejména pokud tyto podmínky a prostředí mohl vývojový tým jen nesnadno replikovat).

Testovací báze

Mezi příklady pracovních produktů, které lze použít jako testovací bázi pro akceptační testování, patří:

- byznysové procesy,
- uživatelské nebo byznysové požadavky,
- regulatorní nařízení, smlouvy a normy,
- případy užití a/nebo uživatelské scénáře,
- systémové požadavky,
- systémová nebo uživatelská dokumentace,
- instalační postupy,
- reporty z analýzy rizik.

Kromě toho lze jako testovací bázi pro odvozování testovacích případů pro provozní akceptační testování použít jeden nebo více následujících pracovních produktů:

- postupy zálohování a obnovy,
- postupy obnovy po havárii,
- nefunkcionální požadavky,
- provozní dokumentace,
- pokyny pro zavádění a instalaci,
- výkonnostní požadavky (cíle),
- databázové balíčky,
- bezpečnostní normy a/nebo příslušná regulatorní nařízení.

Typické objekty testování

Mezi typické objekty pro všechny formy akceptačního testování patří:

- testovaný systém (SUT, system under test),
- konfigurace systému a konfigurační data,
- byznysové procesy pro plně integrovaný systém,
- systémy obnovy a pracoviště schopné převzít bez přerušení funkci původních pracovišť (pro testování kontinuity podnikání a obnovy po havárii),
- procesy provozu a údržby,
- formuláře,
- reporty,
- stávající a konvertovaná výrobní data.

Typické defekty a selhání

Mezi příklady typických defektů pro všechny formy akceptačního testování patří:

- Systémové pracovní toky nesplňují byznysové požadavky nebo uživatelské požadavky.
- Byznysová pravidla nejsou správně implementována do systému.
- Systém nesplňuje smluvní nebo regulatorní požadavky.
- Jsou pozorována nefunkcionální selhání jako například bezpečnostní zranitelnosti, nedostatečná výkonnost pod zátěží nebo nesprávná funkčnost na podporované platformě.

Specifické přístupy a odpovědnosti

Akceptační testování je často zodpovědností zákazníků, byznysových uživatelů, vlastníků produktu nebo provozovatelů systému (ale mohou být do něj zapojeny i další zainteresované osoby).

Akceptační testování je často považováno za poslední úroveň testování v životním cyklu sekvenčního vývoje, ale může se také vyskytovat v jiných fázích, například:

- Akceptační testování krabicového softwaru může být provedeno v okamžiku jeho instalace nebo integrace.
- Akceptační testování vylepšené funkcionality může být provedeno před systémovým testováním.

V případě iterativního vývoje mohou projektové týmy využívat různých forem akceptačního testování během každé iterace a na jejím konci, například na ověření nových vlastností podle jejich akceptačních kritérií nebo pro ověření, zda nové vlastnosti splňují potřeby uživatele. Kromě toho mohou být provedeny alfa a beta testy, buď na konci každé iterace, po dokončení jedné iterace nebo po dokončení série iterací. Rovněž mohou být na konci každé iterace, po dokončení jedné iterace nebo série iterací provedeny uživatelské akceptační testy, provozní akceptační testy, regulatorní nebo smluvní akceptační testy.

2.3 Typy testů

Typ testů je skupina testovacích aktivit zaměřených na testování konkrétních charakteristik softwarového systému nebo jeho části na základě konkrétních testovacích cílů. Mezi tyto cíle může patřit:

- vyhodnocení funkcionálních kvalitativních charakteristik jako jsou kompletnost, správnost a vhodnost,
- vyhodnocení nefunkcionálních kvalitativních charakteristik jako jsou spolehlivost, výkonnost, bezpečnost, kompatibilita a použitelnost,
- vyhodnocení, zda je struktura nebo architektura komponenty či systému správná, úplná a odpovídá specifikacím,
- vyhodnocení dopadu změn, např. potvrzení, že byl defekt opraven (konfirmační testování) a nalezení nechtěných změn chování po změně v softwaru nebo prostředí (regresní testování).

2.3.1 Funkcionální testování

Funkcionální testování systému zahrnuje testy, které ověřují funkce, které by měl systém vykonávat. Funkcionální požadavky mohou být popsány v dokumentech, jako jsou specifikace byznysových požadavků, eposy, uživatelské scénáře, případy užití nebo funkční specifikace (které mohou být i nedokumentované). Funkcionality představují, „co“ by systém měl dělat.

Funkcionální testování by mělo být prováděné ve všech úrovních testování (například testování komponent může být založeno na specifikacích komponent), i když zaměření je na každé úrovni jiné (viz kapitola 2.2).

Funkcionální testování hodnotí chování softwaru, takže mohou být použity techniky návrhu testů černé skříňky s cílem odvodit testovací podmínky a testovací případy pro funkcionalitu komponenty nebo systému (viz kapitola 4.2).

Důkladnost funkcionálního testování může být měřena rozsahem funkcionálního pokrytí. Funkcionální pokrytí je míra, do jaké je určitá funkcionalita prozkoumána testy a vyjadřuje se jako procento pokrytí testy pro daný typ prvku. Například použitím trasovatelnosti mezi testy a funkcionálními požadavky lze vypočítat procento požadavků, které jsou předmětem testů, a potenciálně odhalit mezery v pokrytí.

Návrh a provádění funkcionálních testů může vyžadovat zvláštní dovednosti a znalosti, jako je znalost konkrétního problému v určité oblasti byznysu, který daný software řeší (například software pro geologické modelování pro ropný průmysl).

2.3.2 Nefunkcionální testování

Nefunkcionální testování systému hodnotí charakteristiky systémů a softwaru jako jsou použitelnost, výkonnostní efektivita nebo bezpečnost. Klasifikaci charakteristik kvality softwarových produktů řeší

norma *ČSN ISO/IEC 25010*. Nefunkcionální testování dává odpověď na otázku, „jak“ dobře se systém chová.

Na rozdíl od běžných (mylných) představ může být a často je nefunkcionální testování prováděno ve všech úrovních testování a mělo by proběhnout co nejdříve. Pozdní zjištění nefunkcionálního defektu může být velmi nebezpečné pro úspěch celého projektu.

Pro odvození testovacích podmínek a testovacích případů pro nefunkcionální testování lze použít techniky testování černé skříňky (viz kapitola 4.2). Například pro definování podmínek extrémní zátěže pro testování výkonnosti může být použita technika analýzy hraničních hodnot.

Důkladnost nefunkcionálního testování může být měřena nefunkcionálním pokrytím. Nefunkcionální pokrytí je míra, do jaké je určitý typ nefunkcionálního prvku prozkoumán testy a vyjadřuje se jako procento pokrytí testy pro daný typ prvku. Například použitím trasovatelnosti mezi testy a podporovanými zařízeními pro mobilní aplikace lze vypočítat procento zařízení, která jsou předmětem testů kompatibility, a potenciálně odhalit mezery v pokrytí.

Návrh a provádění nefunkcionálních testů může vyžadovat zvláštní dovednosti a znalosti, jako je znalost principiálních slabých stránek daného designu či technologie (jako jsou bezpečnostní zranitelnosti spojené s určitými programovacími jazyky) nebo konkrétní uživatelské základny (jako jsou osoby uživatelů zdravotnického systému).

Další podrobnosti o testování nefunkcionálních kvalitativních charakteristik jsou uvedeny v učebních osnovách *ISTQB-CTAL-TA*, učebních osnovách *ISTQB-CTAL-TTA*, učebních osnovách *ISTQB-CTAL-SEC* a v dalších specializovaných modulech ISTQB®.

2.3.3 Testování bílé skříňky

Techniky testování bílé skříňky odvozují testy z interních struktur nebo implementace systému. Interní struktura může zahrnovat kód, architekturu, pracovní toky anebo datové toky v rámci systému (viz kapitola 4.3).

Důkladnost testování bílé skříňky může být měřena strukturálním pokrytím. Strukturální pokrytí je míra, do jaké je určitý strukturální prvek prozkoumán testy a vyjadřuje se jako procento prvků daného typu, které byly testováním pokryty.

V úrovni testování komponent je pokrytí kódu definováno procentem kódu komponent, které byly testovány. Může být měřeno ve smyslu různých aspektů tohoto kódu (položek pokrytí), např. procento spustitelných příkazů testovaných v rámci komponenty, nebo procento výstupů rozhodnutí, které byly testovány. Tomuto typu pokrytí se souhrnně říká pokrytí kódu. V úrovni integračního testování komponent může být testování bílé skříňky založeno na prvcích architektury systému, jako jsou rozhraní mezi komponentami, a strukturální pokrytí může být měřeno ve smyslu procenta rozhraní podrobených testování.

Návrh a provádění testů bílé skříňky může vyžadovat zvláštní dovednosti a znalosti, jako je znalost způsobu sestavení kódu, jak se data ukládají (například pro vyhodnocení možných dotazů na databázi) a jak používat nástroje pro měření pokrytí a jak správně interpretovat jejich výsledky.

2.3.4 Testování související se změnami

Po provedení změny v systému (buď kvůli opravě defektu nebo kvůli nové nebo změněné funkcionalitě) je nutno zajistit testování pro potvrzení, že změna skutečně defekt odstranila nebo implementovala danou funkcionalitu správně, a nezpůsobila přitom žádné nepředvídané nežádoucí následky.

- Konfirmační testování: Po opravě defektu může být software testován všemi testy, které v důsledku defektu selhaly, a je třeba je znovu provést na nové verzi softwaru. Software může být také testován zcela novými testy, které pokryjí změny potřebné k opravě defektu. Minimálně však musí být na nové verzi softwaru znovu provedeny kroky, které vedly k selháním způsobeným defektem. Účelem konfirmačního testu je potvrdit, že byl původní defekt úspěšně odstraněn.
- Regresní testování: Stává se, že změna provedená v části kódu, ať už v rámci opravy nebo jiného druhu změny, může náhodně ovlivnit chování jiných částí kódu. Může jít o dopad ve stejné komponentě, v jiných komponentách téhož systému nebo dokonce i v jiném systému nebo systémech. Změny mohou zahrnovat změny prostředí, jako je nová verze operačního systému nebo databázového systému. Takovým nežádoucím vedlejším účinkům se říká regrese. Regresní testy zahrnují testy určené ke zjištění těchto nechtěných vedlejších účinků.

Konfirmační a regresní testování se provádí ve všech úrovních testování. Zejména u iterativně-inkrementálních životních cyklů vývoje softwaru (jako je agilní vývoj) vedou často ke změnám kódu implementace nových vlastností, změny ve stávajících užitných vlastnostech a přepracování kódu (refactoring). Všechny tyto aktivity následně vyžadují testování založené na změnách. Vzhledem k častému průběžnému vývoji a rozšiřování systémů je konfirmační a regresní testování velmi důležité. To je zvlášť významné pro systémy Internetu věcí, kde jsou jednotlivé objekty (například zařízení) velmi často aktualizovány nebo nahrazovány.

Sady regresních testů jsou spouštěny mnohokrát a všeobecně se mění pomalu, proto je regresní testování horkým kandidátem na automatizaci. Automatizace těchto testů by měla začít už v raných fázích projektu (viz kapitola 6).

2.3.5 Typy testů a úrovně testování

Každý výše zmíněný typ testu lze provádět v libovolné úrovni testování. Pro ilustraci budou uvedeny příklady funkcionálních testů, nefunkcionálních testů, testů bílé skříňky a testů souvisejících se změnami napříč všemi úrovněmi testování s využitím příkladu bankovní aplikace, přičemž jako první budou aplikovány funkcionální testy:

- Pro testování komponent se navrhují testy na základě toho, jak je komponenta schopna vypočítat složený úrok.
- Pro integrační testování komponent se testy navrhují na základě toho, jak se informace o účtu získané z uživatelského rozhraní předávají byznysové logice.
- Pro systémové testování se testy navrhují na základě toho, jak majitelé účtů mohou požádat o nějakou formu úvěru ke svému běžnému účtu.
- Pro systémové integrační testování se testy navrhují na základě toho, jak systém využívá externí mikroslužby pro ověření úvěrového hodnocení majitele účtu.
- Pro akceptační testování se testy navrhují na základě toho, jak bankéř zpracovává schválení nebo zamítnutí žádostí o úvěr.

Z pohledu nefunkcionálních charakteristik lze koncipovat testování následujícím způsobem:

- Pro testování komponent se testy výkonu navrhují tak, aby došlo k vyhodnocení počtu cyklů CPU potřebných k provedení složitého výpočtu celkového úroku.
- Pro integrační testování komponent jsou testy bezpečnosti navrženy pro ohodnocení zranitelností rizikem přetečení vyrovnávací paměti díky datům předávaným z uživatelského rozhraní byznysové logice.

- Pro systémové testování jsou testy přenositelnosti navrhovány tak, aby ověřily, zda prezentační vrstva funguje na všech podporovaných prohlížečích a mobilních zařízeních.
- Pro systémové integrační testy jsou testy spolehlivosti navrhovány tak, aby ohodnotily robustnost systému v případě, že mikroslužba úvěrového hodnocení nereaguje.
- Pro akceptační testy jsou testy použitelnosti navrhovány tak, aby vyhodnotily dostupnost rozhraní používaného bankéři pro zpracování úvěrů pro klienty s tělesným postižením.

Pohledem testů bílé skříňky lze koncipovat testování následujícím způsobem:

- Pro testování komponent jsou testy navrhovány tak, aby bylo dosaženo kompletního pokrytí příkazů a rozhodnutí (viz kapitola 4.3) pro všechny komponenty provádějící finanční kalkulace.
- Pro integrační testování komponent jsou testy navrhovány tak, aby ověřily, jak každá obrazovka předává rozhraní prohlížeče data na další obrazovku a do byznysové logiky.
- Pro systémové testování jsou testy navrhovány tak, aby pokrývaly různé posloupnosti webových stránek, které mohou vzniknout při zpracování žádosti o úvěr.
- Pro systémové integrační testování jsou testy navrhovány tak, aby vyzkoušely všechny možné typy dotazů zasílaných do mikroslužby úvěrového hodnocení.
- Pro akceptační testování jsou testy navrhovány tak, aby pokrývaly všechny podporované struktury souborů finančních dat a rozsahy hodnot pro mezibankovní převody.

A konečně z pohledu testování změn lze koncipovat testování následujícím způsobem:

- V úrovni testování komponent se vytváří automatizované regresní testy pro každou komponentu, které jsou pak zařazeny do frameworku průběžné integrace.
- V úrovni integračního testování komponent se navrhují testy s cílem ověření oprav defektů rozhraní. Tyto testy bývají spouštěny obvykle automaticky při současném uložení opraveného kódu do úložiště.
- V úrovni systémového testování se provádějí znovu všechny testy pro daný pracovní tok, v němž se změnila některá obrazovka.
- V úrovni systémového integračního testování se znovu provádějí testy interakce aplikace s mikroslužbou úvěrového hodnocení, a to na denní bázi jako součást průběžného dodávání této mikroslužby.
- V případě akceptačního testování se po opravě defektu zjištěného akceptačním testováním znovu provádějí všechny testy, které selhaly.

I když tato kapitola uvádí příklady všech typů testů pro každou úroveň testování, není nutné mít v každém projektovém kontextu zastoupeny všechny typy testů pro každou úroveň testování. Je však důležité provádět vhodné typy testů v každé úrovni, zejména pak v nejnižší úrovni, kde se daný typ testu může objevit.

2.4 Údržbové testování

Po zavedení do produkčního prostředí musí být software a systémy nadále udržovány. Téměř nevyhnutelné jsou různé změny dodaného softwaru a systémů, buď z důvodu opravy defektu zjištěného za provozu, nebo pro přidání nové funkcionality či pro odstranění nebo změnu funkcionality již dodané. Údržba je potřeba i z důvodu zachování nebo zlepšování nefunkcionálních

kvalitativních charakteristik komponenty nebo systému během jejich životnosti, jako jsou zejména výkonnost, spolehlivost, bezpečnost a přenositelnost.

Pokud je v rámci údržby provedena nějaká změna, musí být provedeno údržbové testování, jednak pro vyhodnocení úspěšnosti provedené změny, a jednak pro kontrolu případných vedlejších účinků (regresí) na části systému, které se neměnily (což je obvykle většina systému). Údržba může zahrnovat plánovaná vydání (release) i neplánovaná vydání (hotfix).

Po provedení změny v rámci údržby může být potřeba zajistit údržbové testování ve více úrovních testování s použitím různých typů testů podle rozsahu změny. Rozsah údržbového testování závisí na:

- míře rizika, v jakém změněná část softwaru komunikuje s jinými komponentami nebo systémy,
- velikosti stávajícího systému,
- rozsahu změny.

2.4.1 Spouštěče údržby

Existuje několik důvodů, proč se provádí údržba softwaru v podobě plánovaných i neplánovaných změn.

Spouštěče údržby lze klasifikovat takto:

- Změny, jako jsou plánované vylepšení funkčnosti (například pro nové vydání), opravy, změny provozního prostředí (například plánovaný upgrade operačního systému nebo databáze), upgrade softwaru COTS a jeho záplaty (patch) pro opravu defektů či zranitelností.
- Migrace (například z jedné platformy na druhou), které mohou vyžadovat provozní testy nového prostředí i změněného softwaru, nebo testy konverze dat, pokud budou migrována data z jiné aplikace do udržovaného systému.
- Vyřazení aplikace z provozu na konci její životnosti.

Při vyřazení aplikace nebo systému může být nutné otestovat migraci nebo archivaci dat, pro která jsou požadována dlouhá období archivace. Dále může být nutné zajistit otestování postupů získávání/obnovy po dlouhodobé archivaci. Kromě toho může být potřeba regresní testování pro ujištění, že všechny funkcionality, které budou nadále používány, fungují.

Pro systémy Internetu věcí může být údržbové testování iniciováno zavedením zcela nové nebo upravené „věci“ jako je hardwarové zařízení nebo softwarová služba do celého systému. Údržbové testování pro takové systémy klade zvláštní důraz na integrační testování na různých úrovních (například na síťové úrovni nebo na úrovni aplikace) a testování aspektů bezpečnosti, zejména těch, které se týkají osobních údajů.

2.4.2 Analýza dopadů pro údržbu

Analýza dopadů hodnotí změny provedené v rámci údržby za účelem stanovení zamýšlených následků i očekávaných a možných vedlejších účinků změny a určení oblastí systému, které budou změnou postiženy. Analýza dopadů může také pomoci určit dopad změny na stávající testy. Vedlejší účinky a postižené oblasti systému musí být testovány z hlediska regresí, ideálně po aktualizaci stávajících testů, které byly změnou postiženy.

Analýzu dopadů lze uskutečnit před provedením změny jako podpůrný prostředek pro rozhodování, zda je změna žádoucí, a to na základě možných následků v ostatních částech systému.

Analýza dopadu může být obtížná, pokud:

- Specifikace (byznysové požadavky, uživatelské scénáře, architektura) jsou zastaralé nebo zcela chybí.
- Testovací případy nejsou dokumentovány nebo jsou zastaralé.
- Obousměrná trasovatelnost mezi testy a testovací bází nebyla udržována.
- Podpora nástrojů je slabá nebo zcela chybí.
- Zúčastněné osoby postrádají znalost dané byznysové domény nebo systému.
- Nebyla věnována dostatečná pozornost udržovatelnosti softwaru během vývoje.

3 Statické testování — 135 minut

Klíčová slova

Revize (přezkoumání), ad hoc revize, revize založená na kontrolním seznamu, dynamické testování, formální revize, neformální revize, inspekce, čtení založené na perspektivě, revize založená na roli, revize založená na scénáři, statická analýza, statické testování, technická revize, předvedení (walkthrough), revize kolegou (buddy-check), průběžné dodávání, průběžné nasazování.

Studijní cíle

3.1 Základy statického testování

FL-3.1.1 (K1) Rozpoznat typy softwarového pracovního produktu, který může být prověřen použitím různých technik statického testování.

FL-3.1.2 (K2) S využitím příkladů popsat význam statického testování.

FL-3.1.3 (K2) Vysvětlit rozdíl mezi statickými a dynamickými technikami s přihlédnutím k cílům, druhům identifikovatelných defektů a významu těchto technik v rámci životního cyklu softwaru.

3.2 Proces revize

FL-3.2.1 (K2) Shrnout činnosti procesu revize pracovního produktu.

FL-3.2.2 (K1) Rozpoznat různé role a odpovědnosti při formální revizi.

FL-3.2.3 (K2) Vysvětlit rozdíly mezi různými typy revizí: neformální revize, předvedení, technická revize a inspekce.

FL-3.2.4 (K3) Použít techniky revize pracovního produktu s cílem najít defekt.

FL-3.2.5 (K2) Vysvětlit faktory, které přispívají k úspěšné revizi.

3.1 Základy statického testování

Na rozdíl od dynamického testování, které vyžaduje spuštění testovaného softwaru, statické testování spoléhá na manuální prověření pracovních produktů (tj. na revizi) nebo na vyhodnocení kódu nebo jiných pracovních produktů s využitím nástroje (tj. statická analýza). Oba typy statického testování posuzují dodaný pracovní produkt nebo kód, aniž by došlo k jeho spuštění.

Statická analýza je důležitá pro bezpečnostně kritické počítačové systémy (např. v softwaru pro letecký, zdravotnický nebo jaderný průmysl), ale stala se také důležitou a běžnou i v jiných oblastech. Statická analýza je například důležitou součástí testování bezpečnosti. Statická analýza je také často součástí systémů pro automatické sestavování a dodávání softwaru, například v agilním vývoji systémy pro průběžné dodávání a průběžné nasazování.

3.1.1 Pracovní produkty, které mohou být prověřeny statickým testováním

Téměř každý pracovní produkt může být prozkoumán pomocí statického testování (revizí a/nebo statickou analýzou), například:

- specifikace, včetně byznysových, funkcionálních a bezpečnostních požadavků,
- eposy, uživatelské scénáře, akceptační kritéria,
- specifikace architektury a návrhu,
- kód,
- testware, včetně testovacích plánů, testovacích případů, testovacích procedur a automatizačních testovacích skriptů,
- uživatelské příručky,
- webové stránky,
- smlouvy, projektové plány, harmonogramy a plány rozpočtů,
- nastavení konfigurací a infrastruktury,
- modely, například diagramy aktivit, které mohou být použity pro testování založené na modelu (viz učební osnovy *ISTQB-CTFL-MBT* a *Kramer 2016*).

Revizi lze použít na jakýkoli pracovní produkt, který dokážou účastníci procesu revize přečíst a porozumět mu. Statickou analýzu lze použít efektivně na jakýkoliv pracovní produkt s formální strukturou (obvykle kód nebo modely), existuje-li odpovídající nástroj. Statickou analýzu lze použít i s nástroji, které hodnotí pracovní produkty napsané v přirozeném jazyce jako jsou požadavky (například pro kontrolu pravopisu, gramatiky a čitelnosti).

3.1.2 Přínosy statického testování

Statické testování poskytuje řadu výhod. Statické testování umožňuje při použití v raných fázích životního cyklu vývoje softwaru odhalení defektů dříve než při dynamickém testování (např. při revizích požadavků nebo návrhů, upřesnění backlogu atd.). Obvykle platí, že je mnohem levnější odstranit defekty zjištěné dříve než defekty zjištěné později v životním cyklu vývoje softwaru. Toto platí zejména ve srovnání s defekty zjištěnými u již nasazeného a používaného softwaru. Použití technik statického testování pro nalezení defektů a k jejich rychlé opravě je pro organizaci téměř vždy mnohem levnější než jeho hledání a odstraňování použitím technik dynamického testování na testovaném objektu. Zejména při zvážení dalších nákladů spojených s aktualizací jiných pracovních produktů a provedením konfirmačních a regresních testů.

Mezi další výhody statického testování patří:

- efektivnější detekce defektů a jejich možná oprava před provedením dynamického testování,
- identifikace defektů, které nejsou snadno odhalitelné při dynamickém testování,
- prevence vzniku defektů v návrhu nebo při kódování odhalením nesrovnalostí, nejednoznačností, rozporů, opomenutí, nepřesností a redundancí v požadavcích,
- zvýšení produktivity vývoje (např. z důvodu lepšího návrhu nebo lépe udržovatelného kódu),
- zkrácení času a snížení nákladů na vývoj,
- zkrácení času a snížení nákladů na testování,
- snížení celkových nákladů na kvalitu po dobu životnosti softwaru z důvodu snížení počtu selhání objevených později v životním cyklu nebo až po dodání do provozu,
- zlepšení komunikace mezi členy týmu v případě jejich účasti při revizích.

3.1.3 Rozdíly mezi statickým a dynamickým testováním

Statické a dynamické testování může mít tytéž cíle (viz kapitola 1.1.1), například posouzení kvality pracovních produktů a včasná identifikace defektů. Statické a dynamické testování se vzájemně doplňují tím, že nalézají různé typy defektů.

Hlavní rozdíl spočívá v tom, že statické testování zjistí defekty v pracovních produktech přímo, zatímco dynamické testování identifikuje selhání způsobená defekty při spuštění softwaru. Defekt může být přítomen v pracovním produktu po velmi dlouhou dobu, aniž by došlo k selhání. Větev kódu, kde je defekt přítomen, může být spouštěna zřídka nebo je obtížně dosažitelná (např. ošetření výjimek). Proto nemusí být snadné vytvářet a spouštět dynamické testy, které takové selhání vyvolají. Statické testování může najít takový defekt s mnohem menším úsilím.

Dalším rozdílem je, že statické testování může být použito ke zlepšení konzistence a interní kvality pracovních produktů, zatímco dynamické testování se typicky zaměřuje na externě pozorovatelné chování.

Následující seznam zahrnuje typické defekty, které statické testování odhalí snáze a levněji než dynamické testování:

- defekty v požadavcích (např. nesrovnalosti, nejednoznačnosti, rozpory, opomenutí, nepřesnosti a redundance),
- defekty v návrhu (např. neefektivní algoritmy nebo struktury databáze, vysoká duplicita, nízká soudržnost),
- defekty při kódování (např. proměnné s nedefinovanou hodnotou, deklarované, ale nepoužité proměnné, nedosažitelný kód, duplicitní kód),
- odchylky od norem (např. nedostatečné dodržování standardů pro kódování),
- nesprávné specifikace rozhraní (např. použití různých měrných jednotek systémem volajícím než systémem volaným),
- bezpečnostní zranitelnosti (např. náchylnost k přetečení vyrovnávací paměti),
- chybějící části nebo nepřesnosti v trasovatelnosti nebo v pokrytí testovací báze (např. chybějící testy pro některé akceptační kritérium).

Kromě toho lze většinu typů defektů udržovatelnosti nalézt pouze pomocí statického testování (např. nesprávná modularizace, nízká znovupoužitelnost komponent, kód, který je obtížné analyzovat a upravovat bez zavlečení nového defektu).

3.2 Proces revize

Revize se od sebe liší stupněm formálnosti od neformálních až po formální. Neformální revize jsou charakterizovány tím, že není dodržován žádný definovaný proces a nemají formálně zdokumentovaný výstup. Formální revize jsou charakterizovány účastí týmu, dokumentovanými výsledky revize a dokumentovanými postupy pro provádění revize. Formálnost procesu revize souvisí s faktory, jako jsou model životního cyklu vývoje softwaru, vyspělost procesu vývoje, složitost revidovaných pracovních produktů, právní nebo regulatorní požadavky a/nebo potřeby pro prokazování při auditech (tzv. auditních stop).

Zaměření revize závisí na dohodnutých cílech (např. nalezení defektů, pochopení pracovního produktu, vzdělávání účastníků revize jako jsou testeři a noví členové týmu, příp. diskuse a rozhodování získáním shody názorů).

Pro podrobnější popis procesu revize pracovních produktů včetně rolí a technik revize viz norma *ČSN ISO/IEC 20246*.

3.2.1 Proces revize pracovních produktů

Mezi hlavní skupiny činností procesu revize patří:

Plánování

- definice rozsahu zahrnující účel revize, jaké dokumenty nebo části dokumentů budou revidovány a jaké kvalitativní charakteristiky budou vyhodnocovány,
- odhad pracnosti a harmonogram,
- identifikace charakteristik revize jako jsou typ revize včetně rolí, činnosti a kontrolní seznamy,
- výběr lidí k účasti v revizi a rozdělení rolí,
- definice vstupních a výstupních kritérií v případě více formálních typů revizí (např. inspekce),
- kontrola, že jsou splněna vstupní kritéria (pro formálnější typy revizí).

Zahájení revize

- distribuce pracovních produktů (fyzicky nebo elektronicky) a dalších materiálů, například formuláře pro záznam nálezů, kontrolní seznamy a související pracovní produkty,
- vysvětlení rozsahu, cílů, procesů, rolí a pracovních produktů účastníkům revize,
- zodpovězení otázek účastníků revize.

Individuální revize (tj. individuální příprava)

- revize celého nebo části pracovního produktu,
- zaznamenání potenciálních defektů, doporučení a otázek.

Komunikace a analýza problémů

- komunikování identifikovaných potenciálních defektů (např. při revizních schůzkách),
- analýza potenciálních defektů, přiřazení jejich vlastníků a stavů,
- vyhodnocení a dokumentování kvalitativních charakteristik,
- vyhodnocení výsledků revize proti výstupním kritériím s cílem rozhodnout o jejím výsledku (zamítnutí, nutné velké změny, akceptace s případnými menšími změnami).

Opravy a reportování

- vytvoření reportů o defektech pro ta zjištění, které vyžadují změny v pracovním produktu,
- oprava nalezených defektů (obvykle prováděná autorem) v revidovaném pracovním produktu,
- komunikování defektů příslušné osobě nebo týmu (pokud je defekt nalezen v pracovním produktu, který s tím revidovaným souvisí),
- zaznamenání aktualizovaných stavů defektů (při formální revizi), včetně případného odsouhlasení autorem komentáře,
- sběr metrik (pro formálnější typy revize),
- kontrola, že jsou splněna výstupní kritéria (pro formálnější typy revizí),
- akceptace pracovního produktu v případě dosažení výstupních kritérií.

Výsledky revize pracovního produktu se liší podle jejího typu a stupně formálnosti, jak je popsáno v kapitole 3.2.3.

3.2.2 Role a odpovědnosti při formální revizi

Do typické formální revize jsou zahrnuty následující role:

Autor

- Vytváří revidovaný pracovní produkt.
- Opravuje defekty v revidovaném pracovním produktu (podle potřeby).

Manažer (zástupce managementu)

- Je zodpovědný za plánování revizí.
- Rozhoduje o provedení revizí.
- Přiděluje pracovníky, rozpočet a čas.
- Průběžně sleduje efektivitu nákladů.
- Rozhoduje o dalším postupu v případě nedostatečných výsledků.

Prostředník (často označovaný jako moderátor)

- Zajišťuje efektivní průběh revizních schůzek (pokud se konají).
- Je-li to nutné, zprostředkovává vyjednávání mezi různými pohledy.
- Je často osobou, na které závisí úspěch revize.

Vedoucí revize

- Přebírá celkovou odpovědnost za revizi.
- Rozhoduje, kdo bude do revize zapojen, a organizuje, kdy a kde se budou revize konat.

Revidující

- Může se jednat o odborníka na danou oblast, osobu pracující na projektu, osobu zainteresované na daném pracovním produktu a/nebo osobu s konkrétním technickým nebo byznysovým zázemím.
- Identifikuje případné defekty v revidovaném pracovním produktu.

- Může reprezentovat různé pohledy (např. tester, vývojář, uživatel, operátor, byznysový analytik, expert na použitelnost atd.).

Zapisovatel

- Shromažďuje potenciální defekty zjištěné během aktivity individuální revize.
- Zaznamenává nové potenciální defekty, otevřené body a rozhodnutí z revizních schůzek (pokud se konají).

U některých typů revizí může jedna osoba zastupovat více než jednu roli a aktivity spojené s každou rolí se také mohou lišit podle typu revize. Navíc s nástupem nástrojů na podporu procesu revize (zejména při zaznamenávání defektů, otevřených bodů a rozhodnutí) není často potřeba ani fyzický zapisovatel.

Je možno využít i další přídavné role, viz norma *ČSN ISO/IEC 20246*.

3.2.3 Typy revizí

Přestože mohou být revize použity pro různé účely, jedním z hlavních cílů je odhalit defekty. Všechny typy revizí mohou pomoci při zjišťování defektů. Zvolený typ revize by měl vycházet z dalších kritérií výběru, např. dle potřeb projektu, dostupných zdrojů, typu produktu a identifikovaných rizik, byznysové oblasti nebo firemní kultury.

Jeden pracovní produkt může být předmětem více než jednoho typu revize, jejichž pořadí se může lišit. Například neformální revize může předcházet technické revizi, aby se zajistilo, že je pracovní produkt pro technickou revizi připraven.

Druhy revizí popsaných níže mohou být prováděny jako vzájemné revize (peer review), tzn. mohou být prováděny kolegy se stejnou odborností. Typy defektů zjištěných při revizi jsou různé, liší se zejména v závislosti na revidovaném pracovním produktu. Více informací viz kapitola 3.1.3, kde jsou uvedeny příklady defektů, které lze nalézt při revizi v různých pracovních produktech, a *Gilb 1993* pro informace o formálních inspekcích.

Revize mohou být klasifikovány podle různých charakteristik. Následující seznam obsahuje čtyři nejčastější typy revizí a jejich základní charakteristiky.

Neformální revize (např. revize kolegou, párování, párová revize)

- Hlavní účel: detekce potenciálních defektů.
- Možné další účely: tvorba nových nápadů nebo řešení, rychlé řešení menších problémů.
- Není založena na formálním (dokumentovaném) procesu.
- Nemusí obsahovat revizní schůzku.
- Může být prováděna kolegou autora nebo více osobami.
- Výsledky mohou být dokumentovány.
- Užitečnost je různá v závislosti na revidujících.
- Volitelně mohou být využívány kontrolní seznamy.
- Velmi často používaná v agilním vývoji.

Předvedení

- Hlavní účely: zjištění defektů, vylepšení softwarového produktu, zvážení alternativních implementací, zhodnocení shody s normami a specifikacemi.

- Možné další účely: výměna nápadů ohledně použití různých variant technik a stylů zpracování, školení účastníků, dosažení shody.
- Individuální příprava před revizní schůzkou je nepovinná.
- Revizní schůzku obvykle řídí autor pracovního produktu.
- Povinně je přítomen zapisovatel.
- Volitelně mohou být využívány kontrolní seznamy.
- Může mít podobu scénářů, spuštění „nanečisto“ nebo simulací.
- Jsou vytvářeny záznamy o potenciálních defektech a reporty o revizi.
- V praxi se může lišit od poměrně neformálního po velmi formální provedení.

Technická revize

- Hlavní účely: dosažení shody, zjištění potenciálních defektů.
- Možné další účely: vyhodnocení kvality a budování důvěry v pracovní produkt, tvorba nových nápadů, dodání podnětů a motivování autorů pro zlepšování budoucích pracovních produktů, zvážení alternativních implementací.
- Revidujícími by měli být kolegové autora a techničtí odborníci ve stejné nebo i jiné disciplíně.
- Individuální příprava před revizní schůzkou je povinná.
- Revizní schůzka je nepovinná, v ideálním případě je vedena vyškoleným prostředníkem (typicky ne autorem).
- Zapisovatel je povinný, ideálně není totožný s autorem.
- Volitelně mohou být využívány kontrolní seznamy.
- Jsou vytvářeny záznamy o potenciálních defektech a reporty o revizi.

Inspekce

- Hlavní cíle: zjišťování potenciálních defektů, hodnocení kvality a vytváření důvěry v pracovní produkt, předcházení budoucím podobným defektům prostřednictvím vzdělávání autora nebo autorů a analýza kořenových příčin.
- Možné další účely: dodání podnětů a motivování autorů pro zlepšování budoucích pracovních produktů a procesů vývoje softwaru, dosažení shody.
- Je dodržován definovaný proces s formálně zdokumentovanými výstupy založenými na pravidlech a kontrolních seznamech.
- Využívá jasně definované role (viz kapitola 3.2.2), které jsou povinné a mohou zahrnovat i vyčleněného prezentátora (který během revizní schůzky čte pracovní produkt nahlas a často text parafrázuje pomocí vlastních slov).
- Individuální příprava před revizní schůzkou je povinná.
- Revidující jsou buď kolegové autora nebo odborníci z jiných oborů, které se týkají daného pracovního produktu.
- Jsou specifikována vstupní a výstupní kritéria.
- Povinně je přítomen zapisovatel.
- Revizní schůzka je vedena proškoleným prostředníkem (ne autorem).

- Autor nemůže vystupovat jako vedoucí revize, předčítač nebo zapisovatel.
- Mohou být vytvořeny záznamy o potenciálních defektech a reporty o revizi.
- Jsou shromažďovány metriky, které se používají ke zlepšení celého procesu vývoje softwaru, včetně samotného procesu inspekce.

3.2.4 Použití technik revize

Existuje celá řada technik revize, které lze použít při aktivitách konkrétní revize (např. při individuální přípravě) s cílem odhalit defekty. Tyto techniky mohou být použity ve všech typech revizí popsaných výše. Účinnost technik se může lišit v závislosti na typu použité revize. Příklady různých konkrétních revizních technik pro různé typy revizí jsou uvedeny níže.

Ad hoc

Při revizi ad hoc jsou revidujícím poskytnuty malé nebo žádné pokyny ohledně toho, jak by měla být revize vykonávána. Revidující často postupně čtou pracovní produkt a identifikují a dokumentují problémy tak, jak je nalézají. Ad hoc revize je běžně používaná technika, která vyžaduje malou přípravu. Tato technika je velmi závislá na schopnostech revidujících a může vyústit ve vícenásobná hlášení jednoho a téhož problému různými revidujícími.

Revize založená na kontrolním seznamu

Revize založená na kontrolním seznamu je systematickou technikou, při níž revidující zjišťují problémy pomocí kontrolních seznamů, které jsou distribuovány při zahájení revize (např. prostředníkem). Kontrolní seznam pro revizi představuje sadu otázek vztažených k potenciálním defektům a je tvořen na základě zkušeností. Kontrolní seznamy by měly být specifické pro daný typ pracovního produktu a měly by být pravidelně revidovány s cílem pokrýt problémy uniklé při předchozích revizích. Hlavní výhodou techniky založené na kontrolním seznamu je systematické pokrytí typických typů defektů. Při individuálních revizích je nutné dbát na to, aby nedocházelo k prostému procházení kontrolního seznamu, ale také k hledání defektů mimo tento kontrolní seznam.

Scénáře a spuštění „nanečisto“

Při revizi založené na scénáři jsou revidujícím poskytnuta strukturovaná pravidla pro čtení pracovního produktu. Revize založená na scénáři pomáhá revidujícím při spuštění pracovního produktu „nanečisto“ na základě jeho předpokládaného využití (pokud je produkt zdokumentován ve vhodném formátu, například ve formě případů užití). Tyto scénáře poskytují revidujícím lepší pokyny pro identifikaci konkrétních typů defektů než prosté položky kontrolního seznamu. Stejně jako při revizích založených na kontrolním seznamu by se revidující neměli omezovat na zdokumentované scénáře z důvodů rizika vynechání jiných typů defektů (např. chybějící vlastnost).

Čtení (revize) založené na perspektivě

Při technice čtení založené na perspektivě (perspective-based reading), podobně jako při revizi založené na rolích, revidující při individuálních revizích nahlíží na revidovaný produkt pohledem různých zainteresovaných stran. Typicky např. pohledem koncového uživatele, marketingu, specialisty na návrh, testera nebo zástupce provozu. Využití těchto různých pohledů zainteresovaných stran vede k větší hloubce v individuální revizi s menším výskytem vícenásobných hlášení stejných problémů mezi revidujícími.

Kromě toho čtení založené na perspektivě také vyžaduje, aby se revidující pokoušeli používat revidovaný pracovní produkt s cílem vytvořit nový produkt, z původního odvozený. Tester by se například pokoušel vytvořit koncept akceptačních testů během čtení specifikace požadavků, aby

zjistil, zda byly zahrnuty všechny potřebné informace. Při čtení založeném na perspektivě se obvykle využívají kontrolní seznamy.

Empirické studie ukázaly, že čtení založené na perspektivě je nejúčinnější obecnou technikou pro revizi požadavků a technických pracovních produktů. Klíčovým faktorem úspěchu je přiměřené zohledňování různých hledisek zainteresovaných stran na základě rizik. Pro více detailů o čtení založeném na perspektivě viz *Shul 2000*, pro více detailů o účinnosti různých typů revizních technik viz *Sauer 2000*.

Revize založená na rolích

Revize založená na rolích je technika, při níž revidující vyhodnocují pracovní produkt z pohledu jednotlivých zainteresovaných rolí. Mezi typické role patří specifické typy koncových uživatelů (zkušený, nezkušený, senior, dítě atd.) a konkrétní role v organizaci (správce uživatelů, správce systému, tester výkonnosti atd.). Platí obdobné principy jako u techniky čtení založené na perspektivě, jsou využívány i podobné role.

3.2.5 Faktory úspěchu při revizi

Pro úspěšnou revizi je třeba zvážit vhodný typ revize a techniky, které budou použité. Kromě toho existuje řada dalších faktorů, které ovlivní výsledek revize.

Mezi faktory úspěchu z hlediska organizace patří:

- Každá revize má jasné cíle, definované při plánování revize a používá je jako měřitelná výstupní kritéria.
- Používají se takové typy revizí, které jsou vhodné k dosažení daných cílů. Musí být také vhodné pro typ a úroveň softwarových pracovních produktů i účastníky revize.
- Jakékoli použité techniky revize jako jsou revize založené na kontrolním seznamu nebo revize založené na rolích, jsou vhodné pro účinnou identifikaci defektů v revidovaném pracovním produktu.
- Využívané kontrolní seznamy se týkají hlavních rizik a jsou aktuální.
- Velké dokumenty jsou sepisovány a revidovány po malých částech. Kvalita je zde zajišťována tím, že autoři získávají včasnou a častou zpětnou vazbu o defektech.
- Účastníci mají dostatek času na přípravu.
- Revize jsou naplánovány s dostatečným předstihem.
- Management podporuje proces revize (např. vyčleněním adekvátního času na revizní aktivity v projektových plánech).
- Revize jsou součástí firemních politik kvality a/nebo testování.

Mezi faktory úspěchu z hlediska lidí patří:

- Do revize jsou zapojeni správní lidé (například lidé s různými dovednostmi či lidé, kteří se na revidovaný produkt budou schopni dívat z různých perspektiv a mohou jej využít jako vstup pro svůj pracovní produkt).
- Testeři jsou vnímáni jako užiteční revidující, kteří jednak přispívají k revizi a zároveň studují pracovní produkt, což jim umožňuje připravit efektivnější testy v předstihu.
- Účastníci věnují dostatek času a pozornosti detailům.
- Revize jsou prováděné po malých částech, takže revidující neztrácejí koncentraci během konkrétních revizí a/nebo během revizních schůzek (pokud se konají).

- Zjištěné defekty jsou respektovány, oceňovány a objektivně zpracovávány.
- Schůzka je správně řízená, takže ji účastníci považují za užitečné využití svého času.
- Revize probíhá v atmosféře důvěry a výsledek není použit pro hodnocení účastníků.
- Účastníci se dokážou vyvarovat nevhodné řeči těla a chování, které může předznamenávat nudu, podráždění či nepřátelství vůči ostatním účastníkům.
- Je poskytnuto vhodné odborné školení, zejména pro formálnější revize jako např. inspekce.
- Učení a zlepšování procesů je v organizaci podporováno.

Pro více informací o úspěšné revizi viz *Gilb 1993*, *Wiegers 2002* a *van Veenendaal 2004*.

4 Techniky testování — 330 minut

Klíčová slova

Technika testování, technika testování černé skříňky, technika testování bílé skříňky, analýza hraničních hodnot, testování založené na kontrolních seznamech, pokrytí, pokrytí rozhodnutí, testování dle rozhodovací tabulky, odhadování chyb, rozdělení tříd ekvivalence, technika testování založená na zkušenostech, průzkumné testování, testování přechodů stavů, pokrytí příkazů, testování případů užití.

Studijní cíle

4.1 Kategorie technik testování

FL-4.1.1 (K2) Vysvětlit charakteristiky, společné znaky a rozdíly mezi technikami testování bílé skříňky, technikami testování černé skříňky a technikami testování založenými na zkušenostech.

4.2 Techniky testování černé skříňky

FL-4.2.1 (K3) Použít techniku rozdělení tříd ekvivalence pro odvození testovacích případů z daných požadavků.

FL-4.2.2 (K3) Použít techniku analýzy hraničních hodnot pro odvození testovacích případů z daných požadavků.

FL-4.2.3 (K3) Použít techniku testování dle rozhodovací tabulky pro odvození testovacích případů z daných požadavků.

FL-4.2.4 (K3) Použít techniku testování přechodů stavů pro odvození testovacích případů z daných požadavků.

FL-4.2.5 (K2) Vysvětlit, jak odvozovat testovací případy z případu užití.

4.3 Techniky testování bílé skříňky

FL-4.3.1 (K2) Vysvětlit techniku pokrytí příkazů.

FL-4.3.2 (K2) Vysvětlit techniku pokrytí rozhodnutí.

FL-4.3.3 (K2) Vysvětlit hodnotu pokrytí příkazů a pokrytí rozhodnutí.

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.

4.1 Kategorie technik testování

Účelem technik testování (včetně těch, které jsou popsány v této kapitole) je přispět k identifikaci testovacích podmínek, testovacích případů a testovacích dat.

Výběr vhodných technik testování, které budou použity, závisí na řadě faktorů, např.:

- složitost komponenty nebo systému,
- regulatorní normy,
- zákaznické nebo smluvní požadavky,
- úrovně a typy rizik,
- dostupná dokumentace,
- znalosti a dovednosti testerů,
- dostupné nástroje,
- čas a rozpočet,
- model životního cyklu vývoje softwaru,
- typy defektů očekávaných pro danou komponentu nebo systém.

Některé techniky jsou lépe použitelné v určitých situacích a pro určité úrovně testů, zatímco jiné mohou být použitelné pro všechny úrovně testů. Při vytváření testovacích případů používají testeři obecně kombinaci různých technik testování pro dosažení nejlepších možných výsledků vzhledem k pracnosti.

Použití technik testování během testovací analýzy, návrhu testů a implementace testů se může vyskytovat od velmi neformálních (malá nebo žádná dokumentace) až po velmi formální. Příslušná úroveň formality závisí na kontextu testování, včetně zralosti procesů vývoje a testování, časových omezeních, bezpečnostních nebo regulatorních požadavcích, znalostech a dovednostech zúčastněných osob a modelu životního cyklu vývoje softwaru.

4.1.2 Kategorie technik testování a jejich charakteristiky

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

Techniky testování černé skříňky (nazývané také jako behaviorální techniky nebo techniky založené na chování) vychází z analýzy příslušné testovací báze (např. formální dokumentace požadavků, specifikací, případů užití, uživatelských scénářů nebo byznysových procesů). Tyto techniky jsou aplikovatelné pro funkcionální i nefunkcionální testování. Techniky testování černé skříňky se zaměřují na vstupy a výstupy testovaného objektu bez hlubší znalosti jeho vnitřní struktury.

Techniky testování bílé skříňky (nazývané také jako strukturální techniky nebo techniky založené na struktuře) vychází z analýzy architektury, detailního návrhu, vnitřní struktury nebo kódu testovaného objektu. Na rozdíl od technik testování černé skříňky se techniky testování bílé skříňky zaměřují na strukturu a způsob zpracování uvnitř testovaného objektu.

Při testování s pomocí technik založených na zkušenostech je využíváno zkušeností vývojářů, testerů a uživatelů při návrhu, implementaci a provedení testů. Tyto techniky jsou často kombinovány také s technikami testování černé a bílé skříňky.

Mezi obecné charakteristiky technik testování černé skříňky patří:

- Testovací podmínky, testovací případy a testovací data jsou odvozeny z testovací báze, která může zahrnovat požadavky, specifikace, případy užití nebo uživatelské scénáře pro daný software.
- Testovací případy mohou být použity k odhalení nesouladů mezi požadavky a jejich implementací, příp. i jejich odchylek.
- Pokrytí je měřeno na základě otestovaných položek testovací báze a techniky použité s touto bází.

Mezi obecné charakteristiky technik testování bílé skříňky patří:

- Testovací podmínky, testovací případy a testovací data jsou odvozeny z testovací báze, která může zahrnovat kód, architekturu softwaru, detailní návrh nebo jakýkoli jiný zdroj informací o struktuře daného softwaru.
- Pokrytí je měřeno na základě testovaných položek v rámci vybrané struktury (např. kód nebo rozhraní) a použité techniky testování.

Mezi obecné charakteristiky technik testování založených na zkušenostech patří:

- Testovací podmínky, testovací případy a testovací data jsou odvozeny z testovací báze, která může zahrnovat znalosti a zkušenosti testerů, vývojářů, uživatelů a dalších zainteresovaných stran. Tyto znalosti a zkušenosti zahrnují pochopení očekávaného použití softwaru, jeho prostředí, pravděpodobných defektů a jejich rozložení (distribuci).

Více informací o technikách testování a jim odpovídajícím metrikám pokrytí lze nalézt v mezinárodní normě *ČSN ISO/IEC/IEEE 29119-4* a dále v *Craig 2002* a *Copeland 2004*.

4.2 Techniky testování černé skříňky

4.2.1 Rozdělení tříd ekvivalence

Rozdělení tříd ekvivalence je technika návrhu testů, pomocí které je možné rozdělit data na oddíly (označovány jako třídy ekvivalence) tak, aby se dal u všech hodnot daného oddílu očekávat stejný způsob zpracování (viz *Kaner 2013* a *Jorgensen 2014*). Existují přitom třídy ekvivalence pro platné i neplatné hodnoty.

- Platné hodnoty jsou takové, které by měly být komponentou nebo systémem akceptovány. Třída ekvivalence, která obsahuje platné hodnoty, se nazývá „platná třída ekvivalence“.
- Neplatné hodnoty jsou takové, které by měly být komponentou nebo systémem odmítnuty. Třída ekvivalence, která obsahuje neplatné hodnoty, se nazývá „neplatná třída ekvivalence“.
- Třídy ekvivalence je možné identifikovat pro jakoukoliv datovou entitu související s testovaným objektem, a to včetně vstupů, výstupů, interních hodnot, hodnot souvisejících s časovým údajem (např. před nebo po události) a parametrů rozhraní (např. integrované komponenty testované během integračního testování).
- V případě potřeby může být jakákoliv třída ekvivalence rozdělena na podtřídy.
- Každá hodnota musí patřit do právě jedné třídy ekvivalence.
- Pokud se v testovacích případech používají neplatné třídy ekvivalence, měla by být každá taková třída testována odděleně, tj. neměla by se kombinovat s jinými neplatnými třídami ekvivalence, aby se zajistilo, že nedojde k maskování (zakrytí) jednotlivých selhání. K maskování selhání může dojít v případě výskytu několika selhání současně, kdy je ale pouze jedno z těchto selhání pozorovatelné, což má pak za následek, že ostatní selhání nejsou zjištěna.

Pro dosažení 100% pokrytí pomocí této techniky musí testovací případy pokrývat všechny identifikované třídy ekvivalence (včetně neplatných), a to použitím minimálně jedné hodnoty z každé třídy. Pokrytí se měří jako poměr počtu tříd ekvivalence testovaných nejméně jednou hodnotou vzhledem k celkovému počtu identifikovaných tříd ekvivalence (obvykle vyjádřeno v procentech). Technika rozdělení tříd ekvivalence je použitelná ve všech úrovních testování.

4.2.2 Analýza hraničních hodnot

Analýza hraničních hodnot (BVA – boundary value analysis) je rozšířením metody rozdělení tříd ekvivalence. Tato technika může být použita pouze za předpokladu, že je třída seřazená a je složena z numerických nebo sekvenčních dat. Minimální a maximální hodnoty (nebo první a poslední hodnoty) třídy jsou její hraniční hodnoty (viz *Beizer 1990*).

Předpokládejme například, že textové pole akceptuje jako vstup jednomístnou kladnou číselnou hodnotu zadávanou pomocí klávesnice s tím, že neceločíselné vstupy není možné zadat. Platný rozsah vstupních hodnot je od 1 do 5 včetně. Existují tedy tři třídy ekvivalence:

- neplatná (příliš malá čísla)
- platná
- neplatná (příliš vysoká čísla).

Pro platnou třídu ekvivalence jsou hraniční hodnoty 1 a 5. Pro neplatnou třídu (příliš vysoká čísla) je hraniční hodnota 6. Pro neplatnou třídu (příliš malá čísla) existuje pouze jedna hraniční hodnota (číslo 0), protože se jedná o třídu s pouze jedním prvkem.

Ve výše uvedeném příkladu určujeme vždy dvě hraniční hodnoty pro každou hranici. Hranice mezi neplatnou (příliš malá čísla) a platnou třídou je definována testovacími hodnotami 0 a 1. Hranice mezi platnou a neplatnou třídou (příliš vysoká čísla) je definována testovacími hodnotami 5 a 6. Některé varianty této techniky určují tři hraniční hodnoty pro každou hranici: hodnotu před hranicí, hodnotu na ní a hodnotu těsně za ní. V předchozím příkladu (s použitím tříbodových hraničních hodnot) jsou dolní hraniční testovací hodnoty 0, 1 a 2 a horní hraniční testovací hodnoty jsou 4, 5 a 6 (viz *Jorgensen 2014*).

Pravděpodobnost nesprávného chování na hranicích tříd ekvivalence je vyšší než chování uvnitř tříd. Je důležité si uvědomit, že jak specifikované (předpokládané), tak implementované (skutečné) hranice mohou být přesunuty na pozice nad nebo pod jejich zamýšlenými pozicemi, mohou být zcela vynechány nebo mohou být doplněny o další nežádoucí hranice. Analýza hraničních hodnot a jejich testování odhalí téměř všechny takové defekty tím, že nutí software ukázat chování z jiné třídy, než je ta, do které by hraniční hodnota měla patřit.

Analýza hraničních hodnot může být použita ve všech úrovních testování. Technika se obecně používá k testování požadavků s číselným oborem hodnot (včetně datumů a časů). Pokrytí hranic tříd ekvivalence se měří jako počet testovaných hraničních hodnot vydělený celkovým počtem identifikovaných hraničních hodnot (obvykle vyjádřeno v procentech).

4.2.3 Testování dle rozhodovací tabulky

Rozhodovací tabulky jsou dobrý způsob, jak zaznamenat složitá byznysová pravidla, která musí být v systému implementována. Při vytváření rozhodovacích tabulek tester určuje podmínky (často vstupy) a výsledné akce (často výstupy) systému. Podmínky a výsledné akce tvoří dvě skupiny řádků tabulky. Řádky s podmínkami jsou obvykle v horní části tabulky a řádky s akcemi v její dolní části. Každý sloupec odpovídá jednomu pravidlu rozhodování, které definuje jedinečnou kombinaci podmínek vedoucí

k provedení akcí spojených s tímto pravidlem. Hodnoty podmínek a akcí se obvykle zobrazují jako booleovské (pravdivé nebo nepravdivé) nebo diskrétní hodnoty (např. červená, zelená, modrá), ale mohou to být také čísla nebo rozsahy čísel. Tyto různé typy podmínek a akcí lze pak nalézt ve stejné společné (rozhodovací) tabulce.

Obecné symboly používané v rozhodovacích tabulkách jsou následující:

Pro podmínky:

- Y znamená, že podmínka je pravdivá (může být také zobrazena jako T nebo 1).
- N znamená, že podmínka je nepravdivá (může být také zobrazena jako F nebo 0).
- — znamená, že na hodnotě podmínky nezáleží (může být také zobrazena jako N/A).

Pro akce:

- X znamená, že akce by měla nastat (může být také zobrazena jako Y nebo T nebo 1).
- Prázdná hodnota znamená, že akce by neměla nastat (může být také zaznačena jedním ze symbolů –, N, F nebo 0).

Úplná rozhodovací tabulka obsahuje tolik sloupců (testovacích případů) tak, aby pokryly každou kombinaci podmínek. Tabulku lze zjednodušit vynecháním sloupců obsahujících kombinace podmínek, které nemohou nastat:

- možné, ale nedosažitelné kombinace podmínek,
- kombinace podmínek, které neovlivňují výsledek.

Více informací o zjednodušování rozhodovací tabulky lze najít v osnovách *ISTQB-CTAL-TA*.

Obecným standardem minimálního pokrytí u testování dle rozhodovací tabulky je mít alespoň jeden testovací případ pro každé rozhodovací pravidlo uvedené v tabulce. To obvykle znamená, že jsou pokryty všechny kombinace podmínek. Pokrytí je měřeno jako poměr počtu rozhodovacích pravidel testovaných alespoň jedním testovacím případem vzhledem k celkovému počtu rozhodovacích pravidel (obvykle vyjádřeno v procentech).

Význam testování dle rozhodovací tabulky spočívá v tom, že pomáhá identifikovat všechny důležité kombinace podmínek, z nichž některé by jinak mohly být přehlédnuty. Pomáhá také při hledání případných nedostatků v požadavcích. Může být použita ve všech situacích, kdy chování softwaru závisí na kombinaci podmínek. Může být použita v jakékoliv úrovni testování.

4.2.4 Testování přechodů stavů

Komponenty nebo systémy mohou na určitou událost reagovat odlišně v závislosti na aktuálních podmínkách nebo předchozím vývoji (např. události, ke kterým došlo během inicializace systému). Předchozí vývoj (historii) lze popsat pomocí tzv. stavů. Diagram přechodů stavů zobrazuje možné stavy softwaru, stejně jako i to, jak software do nějakého stavu vstupuje, vystupuje z něj, a jak mezi nimi přechází. Přechod je iniciován událostí (např. uživatel zadá hodnoty do pole). Stejná událost může vést ke dvěma nebo více různým přechodům ze stejného stavu. Změna stavu může vést k tomu, že software provede nějakou akci (např. zobrazení výsledku výpočtu nebo chybového hlášení).

Tabulka přechodů stavů zobrazuje všechny platné a možné neplatné přechody mezi jednotlivými stavy, stejně tak i události a výsledné akce pro platné přechody. Diagramy přechodů stavů obvykle zobrazují pouze platné přechody (neplatné přechody nejsou zahrnuty).

Testy mohou být navrhovány tak, aby pokryly typickou posloupnost stavů, prověřily všechny stavy, prověřily každý přechod, prověřily specifickou sekvenci přechodů nebo otestovaly neplatné přechody.

Testování přechodů stavů se typicky používá pro aplikace založené na menu a hojně se využívá v oblasti vestavěného softwaru (embedded software). Technika je také vhodná pro modelování byznysového scénáře se specifickými stavy nebo pro testování navigace na obrazovce. Pojem „stav“ je abstraktní – může představovat několik málo řádků kódu nebo naopak celý byznysový proces.

Pokrytí se běžně měří jako počet zjištěných stavů nebo přechodů, které byly otestovány, vydělený celkovým počtem zjištěných stavů nebo přechodů v testovaném objektu (obvykle vyjádřeno v procentech). Další informace o kritériích pokrytí pro testování přechodů stavů lze najít v osnovách *ISTQB-CTAL-TA*.

4.2.5 Testování případů užití

Testy lze odvozovat z případů užití, které jsou specifickým způsobem popisu interakcí mezi aktéry (uživatelé, externí hardware nebo jiné komponenty či systémy) a vyvíjeným softwarem (komponenta nebo systém).

Každý případ užití definuje určité chování, které může subjekt provádět ve spolupráci s jedním nebo více aktéry (*UML 2.5.1 2017*). Lze jej také popsat interakcemi a aktivitami, ale také vstupními podmínkami, výstupními podmínkami a případně i v určitých případech přirozeným jazykem. Interakce mezi aktéry a subjektem mohou vést ke změnám stavu subjektu. Interakce mohou být graficky znázorněny s využitím pracovních toků (workflows), diagramů aktivit nebo modelů byznysových procesů.

Případ užití může zahrnovat i možné odchylky od jeho základního chování, a to i v případě mimořádného chování (v případě výjimky) a v případě zpracování chyb (reakce systému a obnovení po programátorské, aplikační nebo komunikační chybě, která může mít za následek např. chybové hlášení). Testy jsou navrženy tak, aby prověřily definované chování (základní, mimořádné, alternativní a zpracování chyb). Pokrytí lze měřit počtem testovaných chování definovaných pomocí případů užití vzhledem k celkovému počtu všech chování podle případů užití (obvykle vyjádřeno v procentech).

Další informace o kritériích pokrytí pro testování případů užití lze najít v osnovách *ISTQB-CTAL-ATA*.

4.3 Techniky testování bílé skříňky

Testování bílé skříňky je založeno na vnitřní struktuře testovaného objektu. Techniky testování bílé skříňky mohou být použity ve všech úrovních testování, ale dvě techniky související s kódem (popsané v této kapitole) se nejčastěji používají v úrovni testování komponent. Existují pokročilejší techniky používané v některých bezpečnostně kritických prostředích, v mission-critical prostředích a v prostředích s vysokou integritou s cílem dosažení dokonalého pokrytí, avšak ty zde nejsou popisovány. Více informací o těchto technikách lze najít v osnovách *ISTQB-CTAL-TTA*.

4.3.1 Testování a pokrytí příkazů

Testování příkazů prověřuje potenciálně spustitelné příkazy v kódu. Pokrytí se měří jako počet příkazů prověřených testy vydělený celkovým počtem spustitelných příkazů v testovaném objektu (obvykle vyjádřeno v procentech).

4.3.2 Testování a pokrytí rozhodnutí

Testování rozhodnutí prověřuje místa v kódu, kde dochází k rozhodnutí a testuje kód, který je prováděn na základě výsledků těchto rozhodnutí. Za tímto účelem testovací případy vychází z řídicích toků, které vychází z rozhodovacích bodů. Např. pro příkaz IF je jeden tok pro pravdivý výsledek

a jeden pro nepravdivý výsledek, pro příkaz CASE by bylo nutné navrhnout testovací případy pro všechny možné výsledky včetně standardní (default) větve.

Pokrytí se měří jako počet výsledků rozhodnutí prověřených testy vydělený celkovým počtem výsledků rozhodnutí v testovaném objektu (obvykle vyjádřeno v procentech).

4.3.3 Význam testování příkazů a rozhodnutí

V případě dosažení 100% pokrytí příkazů je zajištěno, že všechny spustitelné příkazy v kódu byly testovány alespoň jednou, ale není zajištěno otestování veškeré logiky rozhodování. Ze dvou technik testování bílé skříňky uvedených v těchto osnovách může testování příkazů poskytnout menší míru pokrytí než testování rozhodnutí.

V případě, že je dosaženo 100% pokrytí rozhodnutí, je zajištěno, že byly prověřeny všechny výsledky rozhodnutí (výsledky TRUE i FALSE). To platí i v případě, že neexistuje žádný příkaz provedený pouze při výsledku rozhodnutí FALSE (např. neexistence větve ELSE příkazu IF). Pokrytí příkazů pomáhá odhalit defekty v kódu, který nebyl spuštěn jinými testy. Pokrytí rozhodnutí pomáhá odhalit defekty v kódu v případech, kdy by jiné testy neověřily TRUE i FALSE výsledky rozhodnutí.

Dosažení 100% pokrytí rozhodnutí zaručuje 100% pokrytí příkazů (nikoliv však naopak).

4.4 Techniky testování založené na zkušenostech

Při použití technik testování založených na zkušenostech jsou testovací případy odvozeny z dovedností a intuice testera, ale také na základě jeho zkušeností s podobnými aplikacemi a technologiemi. Tyto techniky mohou být užitečné při definici testů, které by jinými systematičtějšími technikami nebylo snadné identifikovat. V závislosti na přístupu a zkušenostech testera mohou tyto techniky dosahovat velmi různého stupně pokrytí a účinnosti. v některých případech může být obtížné posoudit míru pokrytí a při použití těchto technik nemusí být dokonce vůbec měřitelné.

Běžně používané techniky testování založené na zkušenostech jsou popsány v následujících kapitolách.

4.4.1 Odhadování chyb

Odhadování chyb je technika používaná k předvídání výskytu chyb, defektů a selhání, která je založená na znalostech testera. Mezi tyto znalosti patří:

- jak fungovala aplikace v minulosti,
- jaké typy chyb obvykle vývojáři dělají,
- selhání, která nastala v jiných aplikacích.

Metodický přístup k technice odhadování chyb spočívá ve vytvoření seznamu možných chyb, defektů a selhání, a navržení testů, které by mohly tato selhání a defekty odhalit. Tento seznam lze sestavit na základě zkušeností, údajů o defektech a selháních nebo z obecných znalostí o tom, proč software selhává.

4.4.2 Průzkumné testování

Průzkumné testování je založeno na neformálních testech (nejsou předem definovány), které jsou navrženy, provedeny, zaznamenány a vyhodnoceny dynamicky během provádění testů. Výsledky testů slouží k učení a získání více informací o komponentě nebo systému, a zároveň pro vytvoření testů pro oblasti, které mohou vyžadovat větší rozsah testování.

Průzkumné testování je někdy strukturované do časově a/nebo věcně ohraničených relací, které umožňují snazší monitoring a řízení průběhu testování. Při testování v relacích se průzkumné testování provádí v rámci definovaného časového úseku, kdy tester používá testovací listinu. V ní jsou definovány cíle testování, na jejichž dosažení je toto testování postaveno. Tester může používat dokumenty (sheets) testovací relace pro záznam prováděných kroků a zjištění.

Průzkumné testování je nejvíce užitečné v případě, kdy je specifikace nedostatečná nebo úplně chybí, taktéž v případě výrazného tlaku na testování z hlediska času. Průzkumné testování je také užitečné jako doplněk ostatních formálnějších technik testování.

Průzkumné testování je silně spjato s reaktivními testovacími strategiemi (viz kapitola 5.2.2). Průzkumné testování může zahrnovat použití dalších technik testování černé skříňky, bílé skříňky a technik založených na zkušenostech.

4.4.3 Testování založené na kontrolních seznamech

Při testování založeném na kontrolních seznamech testeři navrhují, implementují a provádějí testy za účelem pokrytí testovacích podmínek uvedených v kontrolním seznamu. V rámci analýzy testeři vytvářejí nový kontrolní seznam nebo rozšiřují již existující, a mohou také použít stávající kontrolní seznam zcela bez úprav. Takové kontrolní seznamy lze sestavit na základě zkušeností, znalostí o tom, co je pro uživatele důležité, nebo na základě pochopení, proč a jak software selhává.

Kontrolní seznamy mohou být vytvořeny jako podpora různých typů testů včetně funkcionálního a nefunkcionálního testování. Pokud chybí detailní testovací případy, může testování založené na kontrolních seznamech poskytnout návod pro testování a zajistit určitý stupeň konzistence. Jelikož se jedná o obecné seznamy (high-level), je pravděpodobné, že se v daném testování může objevit určitá variabilita, což může přispět k většímu pokrytí, ale i také k menší míře opakovatelnosti testu.

5 Management testování — 225 minut

Klíčová slova

Konfigurační management, management defektů, report o defektu, vstupní kritéria, výstupní kritéria, produktové riziko, projektové riziko, riziko, úroveň rizika, testování založené na rizicích, přístup k testování, řízení testování, odhad testování, manažer testování, monitoring testování, plán testování, plánování testování, report o postupu prací při testování, testovací strategie, strategie testování, souhrnný report z testování, tester.

Studijní cíle

5.1 Organizace testování

FL-5.1.1 (K2) Vysvětlit výhody a nevýhody nezávislého testování.
FL-5.1.2 (K1) Určit úkoly manažera testování a testera.

5.2 Plánování a odhadování testování

FL-5.2.1 (K2) Shrnout účel a obsah plánu testování.
FL-5.2.2 (K2) Chápat rozdíly mezi různými strategiemi testování.
FL-5.2.3 (K2) Uvést příklady možných vstupních a výstupních kritérií.
FL-5.2.4 (K3) Použít znalosti o prioritách a závislostech (technických a logických) při vytváření harmonogramu provádění testů pro danou sadu testovacích případů.
FL-5.2.5 (K1) Identifikovat faktory ovlivňující pracnost testování.
FL-5.2.6 (K2) Vysvětlit rozdíl mezi technikou odhadování založenou na metrikách a technikou založenou na expertech.

5.3 Monitoring a řízení testování

FL-5.3.1 (K1) Zapamatovat si metriky používané při testování.
FL-5.3.2 (K2) Shrnout účel, obsah a cílovou skupinu reportů z testování.

5.4 Konfigurační management

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

5.5 Rizika a testování

FL-5.5.1 (K1) Definovat úroveň rizika pomocí pravděpodobnosti a dopadu.
FL-5.5.2 (K2) Rozlišit mezi projektovým a produktovým rizikem.
FL-5.5.3 (K2) Pomocí příkladů popsat, jak může analýza produktových rizik ovlivnit úplnost a rozsah testování.

5.6 Management defektů

FL-5.6.1 (K3) Vytvořit report o defektu, který pokrývá defekt zjištěný během testování.

5.1 Organizace testování

5.1.1 Nezávislost testování

Testovací aktivity mohou být prováděné osobami ve specifické testerské roli, ale mohou je vykonávat i osoby v jiných rolích (např. zákazníci). Určitý stupeň nezávislosti často činí testera efektivnějším při odhalování defektů, což je způsobeno rozdílnými kognitivními předsudky autora a testera (viz kapitola 1.5). Nezávislostí však nelze nahradit dobrou znalost systému a vývojáři mohou efektivně najít mnoho defektů ve svém vlastním kódu.

Stupně nezávislosti testování zahrnují následující (od nízké úrovně nezávislosti až po vysokou úroveň):

- Zcela bez nezávislých testerů – testování zajišťují vývojáři, kteří testují svůj vlastní kód.
- Nezávislí vývojáři nebo testeři v rámci vývojového nebo projektového týmu – testování mohou zajišťovat vývojáři, kteří testují produkty svých kolegů.
- Nezávislý testovací tým nebo oddělení v rámci organizace, které podává reporty projektovému nebo výkonnému managementu.
- Nezávislí testeři z byznysu nebo uživatelské komunity nebo testeři se specializací ve specifických typech testů, jako jsou použitelnost, bezpečnost, výkon, shoda s regulatorními standardy nebo přenositelnost.
- Nezávislí testeři mimo vývojovou organizaci pracující ve stejné lokaci jako vývoj (tzv. on-site, in-house) nebo mimo ni (tzv. off-site, outsourcing).

U většiny typů projektů je vhodné mít více úrovní testů, přičemž některé z těchto úrovní jsou vykonávány nezávislými testery. Vývojáři by se měli podílet na testování (zejména na nižších úrovních), aby měli kontrolu nad kvalitou své vlastní práce.

Způsob implementace nezávislosti testování souvisí s modelem životního cyklu vývoje softwaru. Například v agilním vývoji mohou být testeři součástí vývojového týmu. V některých organizacích používajících agilní přístupy mohou být tito testeři také považováni za součást většího nezávislého testovacího týmu. Kromě toho mohou vlastníci produktů v takových organizacích provádět akceptační testování na konci každé iterace za účelem validace uživatelských scénářů.

Mezi potenciální výhody (+) a nevýhody (-) nezávislosti testování patří:

- (+) Nezávislí testeři dokážou pravděpodobněji rozpoznat různé druhy selhání ve srovnání s vývojáři, a to z důvodu jejich odlišného zázemí, technického nadhledu a předsudků.
- (+) Nezávislý tester může ověřit, napadnout či vyvrátit předpoklady, které měly zainteresované strany v době přípravy specifikace či implementace systému.
- (+) Nezávislí testeři dodavatele mohou otevřeně a objektivně podat zprávu o testovaném systému bez (politického) tlaku společnosti, která jim testování zadala.
- (-) Izolace od vývojového týmu, která může vést k nedostatečné spolupráci, zpožděním při poskytování zpětné vazby vývojovému týmu nebo nepřátelským vztahům s vývojovým týmem.
- (-) Vývojáři mohou ztratit pocit zodpovědnosti za kvalitu.
- (-) Nezávislí testeři mohou být považováni za úzké místo ve vývojovém procesu.
- (-) Nezávislí testeři mohou postrádat některé důležité informace (např. o testovaném objektu).

Mnohé organizace jsou schopny úspěšně dosáhnout přínosů z nezávislosti testování, a přitom eliminovat jeho nevýhody.

5.1.2 Úkoly manažera testování a testera

Tyto učební osnovy definují dvě testerské role – manažer testování a tester. Činnosti a úkoly vykonávané těmito dvěma rolemi závisí na kontextu projektu a produktu a dovednostech lidí v daných rolích a organizaci.

Manažer testování má celkovou odpovědnost za proces testování a úspěšné řízení testovacích činností. Manažerskou roli může v testování vykonávat kvalifikovaný manažer testování anebo projektový manažer, manažer vývoje nebo manažer pro zajištění kvality (QA manažer). Ve větších projektech nebo organizacích může několik testovacích týmů reportovat jednomu manažerovi testování, kouči testování nebo koordinátorovi testů, přičemž každý tým je veden vedoucím testování nebo hlavním testerem.

Mezi typické úkoly manažera testování patří:

- Vypracovat nebo revidovat politiku testování a testovací strategii organizace.
- Plánovat testovací činnosti s ohledem na kontext, pochopení cílů testování a rizik. To může zahrnovat výběr přístupu k testování, odhady pracnosti a nákladů testování, získání zdrojů, definování úrovní testů a testovacích cyklů, a plánování managementu defektů.
- Sestavit a aktualizovat plán(y) testování.
- Koordinovat plán(y) testování s projektovými manažery, vlastníky produktů a dalšími.
- Sladit hledisko testování s dalšími projektovými aktivitami jako je např. plánování integrace.
- Iniciovat analýzu, návrh, implementaci a provádění testů, sledovat průběh a výsledky testů a kontrolovat stav plnění výstupních kritérií, příp. definici hotového, případně napomáhat v aktivitách dokončení testování.
- Připravit a distribuovat reporty o postupu prací při testování a souhrnné reporty z testování na základě získaných informací.
- Přizpůsobit plánování na základě výsledků testů a postupu prací při testování (někdy je toto zdokumentováno v reportech o postupu prací při testování a/nebo v souhrnných reportech z testování pro jiné, v projektu již dokončené, testy) a provést veškerá opatření nezbytná pro řízení testování.
- Podporovat nastavení systému managementu defektů a odpovídajícího konfiguračního managementu testwaru.
- Zavádět vhodné metriky pro měření postupu prací při testování a hodnocení kvality testování a produktu.
- Podporovat výběr a zavádění nástrojů pro podporu procesu testování, včetně poskytování doporučení týkajících se rozpočtu na výběr nástrojů (a případně nákupu nebo podpory), přidělení času a úsilí potřebného na pilotní projekty, a poskytování průběžné podpory při používání nástroje nebo nástrojů.
- Rozhodovat o implementaci testovacího prostředí.
- Propagovat a hájit testery, testovací tým a profesi testování v rámci organizace.
- Rozvíjet dovednosti testerů a podporovat jejich kariérní rozvoj (např. prostřednictvím vzdělávacích plánů, hodnocení výkonu, koučování atd.).

Způsob, jakým se vykonává role manažera testování, se liší v závislosti na životním cyklu vývoje softwaru. Například v agilním vývoji jsou některé z výše uvedených úkolů řešeny v rámci celého týmu. Zejména úkoly, které se týkají testů prováděných v rámci týmu na denní bázi, jsou často vykonávané testerem pracujícím v daném týmu. Některé úkoly, které se týkají současně několika týmů nebo celé organizace, nebo které souvisejí s personálním managementem, mohou provádět manažeři testování mimo vývojový tým (ti se někdy nazývají kouči testování). Pro více informací o managementu procesu testování viz *Black 2009*.

Mezi typické úkoly testera patří:

- Revidovat a přispívat k plánům testování.
- Analyzovat, revidovat a posuzovat požadavky, uživatelské scénáře a akceptační kritéria, specifikace a modely (tj. testovací bázi) s ohledem na testovatelnost.
- Určit a zdokumentovat testovací podmínky a zachytit trasovatelnost mezi testovacími případy, testovacími podmínkami a testovací bází.
- Navrhnout, nastavit a ověřit testovací prostředí, přičemž je často nutná koordinace se systémovými administrátory a správci sítě.
- Navrhnout a implementovat testovací případy a testovací procedury.
- Připravit a/nebo získat testovací data.
- Vytvořit podrobný harmonogram provádění testů.
- Provádět testy, vyhodnocovat výsledky a dokumentovat odchylky od očekávaných výsledků.
- Používat vhodné nástroje pro usnadnění procesu testování.
- Automatizovat testy podle potřeby (může být podporováno vývojářem nebo odborníkem na automatizaci testů).
- Vyhodnotit nefunkcionální charakteristiky, jako jsou výkonnostní efektivita, bezporuchovost (spolehlivost), použitelnost, bezpečnost (security), kompatibilita a přenositelnost.
- Revidovat testy vytvořené ostatními.

Lidé, kteří pracují na testovací analýze, návrhu testů, na realizaci konkrétních typů testů nebo na automatizaci testů, mohou být specialisty v těchto rolích. V závislosti na produktových a projektových rizicích a zvoleném modelu životního cyklu vývoje softwaru mohou různé osoby vykonávat roli testera v různých úrovních testů. Například v úrovni testování komponent a úrovni integračního testování komponent vykonávají roli testera často vývojáři. V úrovni akceptačního testování je role testera často prováděna byznysovými analytiky, odborníky na danou problematiku a uživateli. V úrovni systémového testování a úrovni testování integrace systémů je role testera často prováděna nezávislým testovacím týmem. V úrovni provozního akceptačního testování je role testera často prováděna provozními a/nebo systémovými administrátory systémů.

5.2 Plánování a odhadování testování

5.2.1 Účel a obsah plánu testování

Plán testování popisuje činnosti testování pro projekty vývoje a (často následné) údržbové projekty. Plánování je ovlivněno politikou testování a testovací strategií organizace, životním cyklem vývoje a použitými metodami (viz kapitola 2.1), rozsahem testování, cíli, riziky, omezeními, kritičností, testovatelností a dostupností zdrojů.

S postupem projektu a postupem plánování testování jsou k dispozici další informace a detaily, které je možné zahrnout do plánu testování. Plánování testování je průběžná aktivita a probíhá po celou

dobu životního cyklu produktu (je nutné si uvědomit, že životní cyklus produktu může přesahovat rozsah projektu tak, aby zahrnoval i fázi údržby). Zpětná vazba poskytovaná prostřednictvím testovacích činností by měla být využita k rozpoznání měnících se rizik a zohlednit je při úpravě plánování. Plánování může být zdokumentováno v hlavním plánu testování a v samostatných plánech testování pro specifické úrovně testů (jako jsou systémové testování a akceptační testování) nebo pro jednotlivé typy testů (jako jsou testování použitelnosti a testování výkonu). Plánování testování může zahrnovat následující činnosti, z nichž některé mohou být zdokumentovány v plánu testování:

- určení rozsahu, cílů a rizik testování,
- definování celkového přístupu k testování,
- integraci a koordinaci testovacích činností do činností životního cyklu softwaru,
- rozhodování o tom, co testovat, o lidech a dalších zdrojích potřebných k provádění různých testovacích činností, a o způsobu, jakým budou testovací činnosti prováděny,
- naplánování testovací analýzy, návrhu, implementace, provedení testů a činností pro vyhodnocování, a to buď v konkrétním časovém harmonogramu (např. při sekvenčním vývoji) nebo v kontextu každé iterace (např. při iterativním vývoji),
- výběr metrik pro monitoring a řízení testování,
- určení rozpočtu pro testovací aktivity,
- určení úrovně detailů a struktury testovací dokumentace (např. poskytnutím šablon nebo příkladů dokumentů).

Obsah plánů testování se liší a může přesahovat výše uvedená témata. Vzorové plány testování a příklady obsahu plánu testování lze nalézt v normě *ČSN ISO/IEC/IEEE 29119-3*.

5.2.2 Strategie testování a přístup k testování

Strategie testování poskytuje obecný popis procesu testování, a to obvykle na úrovni produktu nebo celé organizace. Mezi obecné typy strategií testování patří:

- **Analytické**: Tento typ strategie testování je založen na analýze určitého faktoru (např. požadavku nebo rizika). Testování založené na rizicích je příkladem analytického přístupu, kdy jsou testy navrženy a prioritizovány na základě úrovně rizika.
- **Založené na modelu**: U tohoto typu strategie testování jsou testy navrženy na základě určitého modelu požadovaného aspektu produktu, jako jsou funkce, byznysový proces, vnitřní struktura nebo nefunkcionální charakteristika (např. spolehlivost). Příkladem mohou být modely byznysových procesů, stavové modely a modely růstu spolehlivosti.
- **Metodické**: Tento typ strategie testování spočívá v systematickém použití některé předem definované sady testů nebo testovacích podmínek, jako jsou taxonomie běžných nebo pravděpodobných typů selhání, seznam důležitých charakteristik kvality nebo organizační standardy pro jednotný vzhled (look-and-feel) pro mobilní aplikace nebo webové stránky.
- **Založené na shodě s procesem** (nebo založené na shodě s normou): Tento typ strategie testování zahrnuje analýzu, návrh a provádění testů založených na externích pravidlech a normách. Tato pravidla mohou být definována normami specifickými pro dané odvětví, dokumentací procesů, důslednou identifikací a používáním testovací báze nebo jakýmkoliv procesem nebo normou, kterou ukládá organizace nebo je organizaci ukládána.
- **Řízené** (neboli konzultační): Tento typ strategie testování je založený především na konzultacích, vedení nebo pokynech zainteresovaných stran, odborníků z oboru nebo technických expertů, kteří mohou být mimo testovací tým nebo i mimo samotnou organizaci.

- **Regresně averzní**: Tento typ strategie testování je motivován snahou vyhnout se regresi stávajících schopností produktu a zahrnuje opětovné použití stávajícího testwaru (zejména testovacích případů a testovacích dat) a rozsáhlou automatizaci regresních testů a standardních testovacích sad.
- **Reaktivní:** U tohoto typu strategie testování je průběh testování reakcí na chování testované komponenty nebo systému. Zohledňuje události, ke kterým dochází během provádění testu místo toho, aby bylo testování předem naplánováno (jako je tomu u předchozích strategií). Testy jsou navrhovány, implementovány a mohou být prováděny v okamžité reakci na znalosti získané z výsledků předchozích testů. Průzkumné testování je obvyklou technikou používanou v reaktivních strategiích.

Vhodná strategie testování je často tvořena kombinací výše uvedených typů. Například testování založené na rizicích (analytická strategie) může být kombinováno s průzkumným testováním (reaktivní strategie) – vzájemně se doplňují a při společném použití mohou zajistit účinnější testování.

Zatímco strategie testování poskytuje obecný popis procesu testování, přístup k testování přizpůsobuje tuto strategii pro konkrétní projekt nebo vydání. Přístup k testování tvoří základ pro výběr technik testování, úrovní testů a typů testů, a pro definici vstupních a výstupních kritérií (případně pro definici připravenosti a definici hotového). Přizpůsobení obecné strategie testování je založeno na rozhodnutích ovlivněných složitostí a cíli projektu, povahou vyvíjeného produktu a analýzou produktových rizik. Zvolený přístup závisí na kontextu a může vzít v úvahu faktory jako jsou rizika, bezpečí (safety), dostupné zdroje a dovednosti, technologie, povahu systému (např. systém vytvořený na zakázku versus krabicový software), cíle testování a regulatorní předpisy.

5.2.3 Vstupní kritéria a výstupní kritéria (definice připravenosti a definice hotového)

Pro zajištění efektivního řízení kvality softwaru (a také testování) se obecně doporučuje stanovit kritéria definující, kdy má začít daná testovací činnost a kdy je tato činnost ukončena. Vstupní kritéria (v agilním vývoji typicky nazývaná definice připravenosti) určují vstupní podmínky pro realizaci dané testovací činnosti. Pokud nejsou splněna vstupní kritéria, je pravděpodobné, že tato činnost bude obtížnější, časově náročnější, nákladnější a riskantnější. Výstupní kritéria (v agilním vývoji typicky nazývaná definice hotového) určují podmínky, kterých musí být dosaženo, aby bylo možné prohlásit úroveň testů nebo sadu testů za dokončenou. Vstupní a výstupní kritéria by měla být definována pro každou úroveň testů a typ testu. Tato kritéria se pak mohou lišit v závislosti na stanovených cílech testování.

Mezi typická vstupní kritéria patří:

- dostupnost testovatelných požadavků, uživatelských scénářů a/nebo modelů (např. při použití strategie založené na modelu),
- dostupnost položek testování, které splnily výstupní kritéria pro předchozí úrovně testů,
- dostupnost testovacího prostředí,
- dostupnost potřebných testovacích nástrojů,
- dostupnost testovacích dat a dalších potřebných zdrojů.

Mezi typická výstupní kritéria patří:

- provedení plánovaných testů,
- dosažení stanovené úrovně pokrytí (např. požadavků, uživatelských scénářů, akceptačních kritérií, rizik, kódu),

- počet nevyřešených defektů je v rámci dohodnutého limitu,
- počet odhadovaných zbývajících defektů je dostatečně nízký,
- hodnocené úrovně spolehlivosti, výkonnostní efektivity, použitelnosti, bezpečnosti a dalších relevantních charakteristik kvality jsou dostačující.

Běžně se stává, že testovací aktivity jsou zredukovány či ukončeny dokonce i bez splnění výstupních kritérií, obvykle z důvodu vyčerpání rozpočtu, vypršení plánovaného času a/nebo z důvodu tlaku pro včasné uvedení produktu na trh. Za takových okolností může být ukončení testování přijatelné, pokud na projektu zainteresované strany a byznysoví vlastníci přezkoumají a přijmou rizika spojená s nasazením produktu do produkce bez dalších testů.

5.2.4 Harmonogram provádění testů

Po vytvoření testovacích případů, testovacích procedur (některé testovací procedury mohou být automatizované) a jejich sestavení do testovacích sad, mohou být tyto testovací sady uspořádány do harmonogramu provádění testů definujícím pořadí, v jakém mají být prováděny. Harmonogram provádění testů by měl zohledňovat faktory jako jsou stanovení priorit, závislosti, konfirmační testy, regresní testy a nejúčinnější posloupnost pro provádění testů.

V ideálním případě by testovací případy měly být seřazeny na základě priorit. Testovací případy s nejvyšší prioritou jsou pak obvykle provedeny nejdříve. Toto však neplatí v případech, kdy mezi testovací případy nebo testovanými vlastnostmi (systému) existují závislosti. Pokud testovací případ s vyšší prioritou závisí na testovacím případě s nižší prioritou, musí být nejdříve proveden testovací případ s nižší prioritou. Podobně, pokud existují závislosti napříč testovacími případy, je nutné je seřadit vhodným způsobem bez ohledu na jejich relativní priority. Zároveň je třeba stanovit priority pro konfirmační a regresní testy na základě důležitosti rychlé zpětné vazby ke změnám, přičemž i zde mohou existovat závislosti.

V určitých případech je možno provést různé sekvence testů s rozličnými úrovněmi účinnosti. V takových případech musí docházet ke kompromisům mezi účinností provedení testů a dodržením stanovených priorit.

5.2.5 Faktory ovlivňující pracnost testování

Odhad pracnosti testování zahrnuje očekávané množství práce, které bude zapotřebí k dosažení cílů testování pro konkrétní projekt, vydání nebo iteraci. Mezi faktory ovlivňující pracnost testování patří výsledky testů a charakteristiky produktu, vývojového procesu a lidí:

Charakteristiky produktu

- rizika spojená s produktem,
- kvalita testovací báze,
- velikost produktu**,**
- komplexita oblasti, pro kterou je produkt určen,
- požadavky na charakteristiky kvality (např. bezpečnost, spolehlivost),
- požadovaná úroveň detailu testovací dokumentace,
- požadavky na dodržování právních a regulatorních předpisů.

Charakteristiky vývojového procesu

- stabilita a zralost organizace,
- použitý vývojový model,

- přístup k testování,
- použité nástroje,
- proces testování,
- časový tlak.

Charakteristiky lidí

- dovednosti a zkušenosti zúčastněných osob, zejména s podobnými projekty a produkty (např. znalost domény),
- soudržnost týmu a jeho vedení.

Výsledky testů

- počet a závažnost zjištěných defektů,
- požadované množství oprav.

5.2.6 Techniky pro odhadování testování

Existuje celá řada technik odhadování, které slouží k určení pracnosti potřebné pro přiměřené testování. Dvě nejčastěji používané techniky jsou:

- Technika založená na metrikách: odhadování pracnosti testování založené na metrikách z předchozích podobných projektů nebo na základě typických hodnot.
- Technika založená na expertech: odhadování pracnosti testování na základě zkušeností vlastníků testovacích úkolů nebo odborníků.

V agilním vývoji jsou příkladem přístupů založených na metrikách grafy burn-down, kdy se zbývající pracnost nejprve zaznamená a reportuje. Následně se tato informace používá jako podklad pro určení rychlosti týmu (velocity) a tím lze pak odhadnout množství práce, kterou tým zvládne v příští iteraci. Příkladem přístupu založeného na expertech v agilním vývoji je plánovací poker, kdy členové týmu na základě svých zkušeností odhadují pracnost pro dodání produktové vlastnosti (viz učební osnovy *ISTQB-CTFL-AT*).

U sekvenčních projektů je příkladem přístupu založeného na metrikách odhadování pomocí modelů odstraňování defektů. V těchto případech se zaznamenává a reportuje množství defektů a čas na jejich odstranění, což pak poskytuje podklad pro odhady budoucích projektů podobného charakteru. Příkladem přístupu založeného na expertech u sekvenčních projektů je technika odhadů Wideband Delphi, u které skupina odborníků poskytují odhady založené na svých zkušenostech (viz učební osnovy *ISTQB-CTAL-TM*).

5.3 Monitoring a řízení testování

Účelem monitoringu testování je shromažďování informací, poskytování zpětné vazby o testovacích činnostech a celkově jejich zviditelňování. Informace, které je třeba monitorovat, mohou být shromažďovány ručně nebo automaticky. Získané informace by měly být použity k posouzení postupu prací při testování a ke stanovení, zda jsou splněna výstupní kritéria definovaná pro testování (jako je splnění cílů pro pokrytí produktových rizik, požadavků nebo akceptačních kritérií). V případě agilních projektů se pak jedná o splnění testovacích úkolů souvisejících s definicí hotového.

Řízení testování označuje jakékoliv řídicí nebo nápravné akce, které byly v důsledku shromážděných informací a metrik přijaty (případně reportovány). Akce mohou pokrývat jakoukoli testovací činnost a mohou ovlivnit jakoukoli jinou aktivitu životního cyklu softwaru.

Příklady akcí souvisejících s řízením testování zahrnují:

- přehodnocení priorit testů při výskytu identifikovaného rizika (např. zpoždění dodávky softwaru),
- změna harmonogramu testů v důsledku dostupnosti nebo nedostupnosti testovacího prostředí nebo jiných zdrojů,
- přehodnocení, zda testovaná položka splňuje vstupní nebo výstupní kritéria v důsledku přepracování.

5.3.1 Metriky používané v testování

Metriky lze shromažďovat v průběhu a na konci testovacích činností za účelem posouzení:

- postupu proti plánovanému harmonogramu a rozpočtu,
- aktuální kvality testovaného objektu,
- vhodnosti přístupu k testování,
- efektivity testovacích činností s ohledem na stanovené cíle.

Mezi obecné testovací metriky patří:

- procentní podíl práce, která již byla vykonaná při přípravě testovacích případů proti plánu (nebo procentní podíl implementovaných testovacích případů proti plánovanému stavu),
- procentní podíl práce, která byla již vykonaná při přípravě testovacího prostředí proti plánu,
- postup v provádění testovacích případů (např. počet spuštěných testovacích případů k počtu nespuštěných, počet testovacích případů, které prošly/selhaly, a/nebo počet testovacích podmínek, které prošly/selhaly),
- informace o defektech (např. hustota defektů, zjištěné a opravené defekty, míra selhání a výsledky konfirmačních testů),
- míra pokrytí požadavků, uživatelských scénářů, akceptačních kritérií, rizik nebo kódu,
- dokončení úkolů, alokace a využití zdrojů a úsilí,
- náklady na testování, včetně poměru nákladů a přínosů zjištění dalšího defektu nebo poměru nákladů a přínosů provedení dalšího testu.

5.3.2 Účel, obsah a cílová skupina reportů z testování

Účelem reportování je shrnout a sdělit informace o testovacích činnostech jak v jejich průběhu, tak i po jejich dokončení (např. pro danou úroveň testů). Report z testování připravený během testovací činnosti je označován jako report o postupu prací při testování, zatímco report připravený na konci testovací činnosti bývá označována jako souhrnný report z testování.

Během monitoringu a řízení testování je úkolem manažera testování pravidelně podávat reporty o postupu prací při testování zainteresovaným stranám. Vedle obsahu společného pro reporty o postupu prací při testování a souhrnné reporty z testování, mohou reporty o postupu testování také obsahovat:

- stav jednotlivých testovacích činností a postup proti plánu testování,
- faktory, které brání postupu prací,
- shrnutí testování plánovaného pro další reportovací období,
- informace o kvalitě testovaného objektu.

Při dosažení výstupních kritérií připravuje manažer testování souhrnný report z testování. Tento report obsahuje shrnutí provedených testů na základě posledního reportu o postupu prací při testování a jakékoli další relevantní informace.

Typické souhrnné reporty z testování mohou zahrnovat:

- shrnutí provedených testů,
- informace o tom, co se dělo během testovacího období,
- odchylky od plánu, včetně odchylek v harmonogramu, trvání nebo pracnosti testovacích činností,
- stav testování a informace o kvalitě produktu s ohledem na výstupní kritéria nebo definici hotového,
- faktory, které zablokovaly nebo stále blokují další postup testování,
- metriky týkající se defektů, testovacích případů, pokrytí testů, postupu v činnostech a čerpání zdrojů (viz např. kapitola 5.3.1),
- reziduální (zbytková) rizika (viz kapitola 5.5),
- pracovní produkty z testování vhodné pro opětovné použití.

Obsah reportu z testování se liší v závislosti na projektu, organizačních požadavcích a životním cyklu vývoje softwaru. Například u komplexního projektu s mnoha zainteresovanými stranami nebo u projektu podléhajícímu regulatorním nařízením může být vyžadováno podrobnější a důkladnější reportování než v projektech zajišťujících drobné aktualizace softwaru. Naopak v agilním vývoji může být například report o postupu testování začleněn do tabulek úkolů (task boards), přehledů defektů nebo grafů burn-down, které mohou být diskutovány během denních schůzek (stand-up meeting), viz učební osnovy *ISTQB-CTFL-AT*.

Reporty z testování by měly být přizpůsobeny nejen kontextu projektu, ale i jejich příjemcům (cílové skupině). Druh a množství informací, které by měly být zahrnuty pro technicky zaměřenou cílovou skupinu nebo testovací tým, se může lišit od toho, co by mělo být součástí souhrnného reportu pro management. V prvním případě mohou být důležité informace o typech defektů a jejich trendech, v druhém případě může být vhodnější poskytnout obecný (high-level) report (např. souhrnný stav defektů podle priorit, aktuální stav čerpání rozpočtu, harmonogram a testovací podmínky, které prošly/neprošly, resp. nebyly testované).

Norma *ČSN ISO/IEC/IEEE 29119-3* popisuje dva typy reportů z testování: report o postupu testování a report o dokončení testování (které v těchto učebních osnovách nazýváme souhrnné reporty z testování), a zároveň poskytuje jejich doporučenou strukturu a příklady.

5.4 Konfigurační management

Účelem konfiguračního managementu je vytvořit a udržovat integritu komponenty nebo systému, testwaru a jejich vzájemných vztahů během projektového a produktového životního cyklu.

Konfigurační management by měl z pohledu podpory kvalitního testování zajistit následující:

- Všechny položky testování jsou jednoznačně identifikovány, jejich verze jsou řízeny, změny jsou sledovány a je mezi nimi definován vzájemný vztah.
- Všechny položky testwaru jsou jednoznačně identifikovány, jejich verze jsou řízeny a změny jsou sledovány. Zároveň je definován vzájemný vztah nejen mezi jednotlivými položkami testwaru, ale i vztah s verzemi položek testování tak, aby mohla být udržována trasovatelnost během celého procesu testování.

- Všechny identifikované dokumenty a softwarové položky jsou jednoznačně odkazovány v dokumentaci k testování.

Během plánování testování by měly být identifikovány a implementovány postupy konfiguračního managementu a infrastruktura (nástroje).

5.5 Rizika a testování

5.5.1 Definice rizika

Riziko je možná událost v budoucnu, která má negativní důsledky. Úroveň rizika je určena pravděpodobností, že událost nastane, a dopadem (škodou), pokud událost skutečně nastane.

5.5.2 Produktová a projektová rizika

Produktové riziko se týká možnosti, že pracovní produkt (např. specifikace, komponenta, systém nebo test) nemusí uspokojit oprávněné potřeby jeho uživatelů a/nebo zainteresovaných stran. Pokud jsou produktová rizika spojena se specifickými charakteristikami kvality daného produktu (např. funkcionální vhodnost, spolehlivost, výkonnostní efektivita, použitelnost, bezpečnost, kompatibilita, udržovatelnost a přenositelnost), pak se tato produktová rizika označují také jako rizika kvality. Příklady produktových rizik zahrnují:

- Software nemusí plnit jeho zamýšlené funkce podle specifikace.
- Software nemusí plnit jeho zamýšlené funkce podle potřeb uživatelů, zákazníků a/nebo zúčastněných stran.
- Architektura systému nemusí plně vyhovovat některým nefunkcionálním požadavkům.
- Za určitých okolností může být nesprávně provedený konkrétní výpočet.
- Řídicí struktury smyčky mohou být nakódovány nesprávně.
- Doba odezvy může být nedostatečná pro systém zpracování transakcí.
- Uživatelský prožitek (user experience, UX) nemusí naplňovat očekávání od produktu.

Projektová rizika se týkají situací, které, pokud by nastaly, by mohly mít negativní dopad na dosažení stanovených cílů projektu. Mezi příklady projektových rizik patří:

- projektové problémy
    - zpoždění při dodání, dokončení úkolů nebo plnění výstupních kritérií nebo definice hotového,
    - nepřesné odhady, převod finančních prostředků na projekty s vyšší prioritou nebo obecné snížení nákladů v celé organizaci mohou mít za následek nedostatečné financování,
    - změny v pozdních fázích projektu mohou vést k podstatným dodatečným nákladům na přepracování,
- organizační problémy
    - nedostatečné dovednosti, školení a nedostatečný počet pracovníků,
    - osobní problémy mohou způsobit konflikty a narušit mezilidské vztahy,
    - uživatelé, zástupci byznysu nebo odborníci nemusí být k dispozici kvůli prioritám daných byznysem,
- politicko-sociální problémy

  - testeři nemusí přiměřeně sdělovat své potřeby a/nebo výsledky testů,
  - vývojáři a/nebo testeři nemusí zohlednit informace získané při testování a revizích (např. nebudou zlepšovat postupy vývoje a testování),
  - může existovat nevhodný postoj k testování nebo nepřiměřená očekávání od testování (např. nedocenění hodnoty zjištění defektů v průběhu testování),
- technické problémy
  - požadavky nemusí být dostatečně dobře definovány,
  - požadavky nemusí být splněny vzhledem k existujícím omezením,
  - testovací prostředí nemusí být připraveno včas,
  - konverze dat, plánování migrace a jejich podpora nástroji může být zpožděna,
  - nedostatky v procesu vývoje mohou mít vliv na konzistenci nebo kvalitu projektových pracovních produktů, jako jsou návrh, kód, konfigurace, testovací data a testovací případy,
  - špatný management defektů a podobné problémy mohou mít za následek nashromážděné defekty a další prvky technologického dluhu,
- problémy s dodavateli
  - třetí strana nemusí dodat potřebný produkt nebo službu nebo může dokonce zkrachovat,
  - smluvní problémy mohou způsobit problémy projektu.

Projektová rizika mohou ovlivnit jak vývojové, tak i testovací činnosti. V některých případech jsou za řešení všech projektových rizik zodpovědní projektoví manažeři, ale není nijak neobvyklé, když je tato zodpovědnost delegována manažerům testování.

5.5.3 Testování založené na rizicích a kvalita produktu

Velmi často využíváme rizika pro zacílení aktivit testování. Používají se k rozhodnutí o tom, kde a kdy začít s testováním a k identifikaci oblastí, které vyžadují větší pozornost. Testování slouží ke snížení pravděpodobnosti vzniku nežádoucí události nebo ke snížení jejího dopadu. Testování se používá jako aktivita pro zmírnění rizik, k poskytnutí informace o zjištěných rizicích a také k poskytnutí informace ohledně zbytkových (nevyřešených) rizik.

Přístup k testování založený na rizicích poskytuje proaktivní příležitosti ke snížení úrovně produktových rizik. Vyžaduje analýzu produktových rizik, která zahrnuje identifikaci produktových rizik a posouzení pravděpodobnosti a dopadu každého rizika. Získané informace o produktových rizicích se využívají k správnému nasměrování plánování testování, specifikaci, přípravě a provádění testovacích případů, a monitoringu a řízení testování. Včasná analýza produktových rizik přispívá k úspěchu projektu.

Při přístupu založeném na rizicích se výsledky analýzy produktových rizik používají k:

- určení technik testování, které budou použity,
- určení konkrétních úrovní a typů testů, které je třeba provést (např. bezpečnostní testování, testování přístupnosti),
- určení rozsahu testování, které se má provést,
- prioritizaci testování s cílem nalezení kritických závad co nejdříve,

- určení, zda mohou být ke snížení rizika použity jiné aktivity než testování (např. zajištění školení pro nezkušené návrháře).

Testování založené na rizicích staví při analýze projektových rizik na kolektivních znalostech a poznatcích zainteresovaných stran projektu. Pro minimalizaci pravděpodobnosti selhání produktu, zajišťují činnosti řízení rizik systematický přístup pro:

- analýzu (a pravidelná přehodnocení) toho, co se může pokazit (rizika),
- určení rizik, které je nutné řešit,
- zavedení opatření ke zmírnění těchto rizik,
- vytvoření pohotovostních plánů (contingency plans) na řešení rizik, pokud skutečně nastanou.

Testování může identifikovat nová rizika, pomáhá určit, která rizika by měla být zmírněna, a také přispívá ke snížení nejistoty ohledně rizik.

5.6 Management defektů

Jelikož jedním z cílů testování je nalezení defektů, musí být defekty zjištěné během testování zaznamenávány. Způsob, jakým jsou defekty zaznamenávány, se může lišit v závislosti na kontextu testované komponenty nebo systému, úrovni testování a modelu životního cyklu vývoje softwaru. Každý zjištěný defekt by měl být přezkoumán, a měl by být sledován po celou dobu od jeho objevení a klasifikace až po jeho vyřešení (např. oprava defektu a úspěšné konfirmační testování daného řešení, odložení do následného vydání, přijetí jako trvalé omezení produktu apod.). Z toho důvodu musí organizace vytvořit proces managementu defektů, jehož cílem je vyřešení všech defektů, a který zahrnuje pracovní tok (workflow) a pravidla pro klasifikaci. Tento proces musí být odsouhlasený všemi účastníky managementu defektů, včetně architektů, návrhářů, vývojářů, testerů a vlastníků produktů. V některých organizacích může být zaznamenávání a sledování defektů velmi neformální.

Během procesu managementu defektů se mohou některé reporty o defektech ukázat jako falešně-pozitivní, nikoliv jako skutečná selhání způsobená defekty. Například může dojít k selhání testu v případě výpadku sítě nebo vypršení časového limitu. Toto chování nevyplývá z defektu v testovaném objektu, ale jedná se o anomálii, která musí být dále zkoumána. Testeři by se měli snažit minimalizovat počet falešně-pozitivních reportů hlášených jako defekty.

Defekty mohou být hlášeny během kódování, statické analýzy, revizí nebo i během dynamického testování či používání softwarového produktu. Defekty mohou být hlášeny v případě problémů v kódu nebo v běžících systémech nebo v jakémkoli typu dokumentace včetně požadavků, uživatelských scénářů a akceptačních kritérií, vývojových dokumentů, testovacích dokumentů, uživatelských příruček nebo instalačních návodů. V zájmu efektivního a účinného procesu managementu defektů mohou organizace definovat standardy pro atributy, klasifikaci a pracovní toky defektů.

Reporty o defektu mají obvykle následující cíle:

- Poskytnout vývojářům a dalším zainteresovaným stranám informace o všech nepříznivých událostech tak, aby mohli určit konkrétní dopady, izolovat problém pomocí jednoduchého reprodukčního testu a opravit potenciální defekt(y) podle potřeby nebo jinak daný problém vyřešit.
- Poskytnout manažerům testování prostředek ke sledování kvality pracovního produktu a dopadů na testování (např. situace, kdy je nahlášeno mnoho defektů a testeři tak tráví

velké množství času jejich reportováním a prováděním konfirmačních testů namísto provádění jiných testů).

- Navrhnout nápady pro zlepšování vývojových a testovacích procesů.

Report o defektu podaný během dynamického testování obvykle obsahuje tyto informace:

- identifikátor,
- název a krátké shrnutí zjištěného defektu,
- datum reportu o defektu, autor a jeho organizační zařazení,
- identifikace položky testovaní (testovaná konfigurační položka) a prostředí,
- fáze životního cyklu vývoje, ve které/kterých byl defekt pozorován,
- popis defektu umožňující reprodukci a vyřešení, včetně protokolů (logs), záloh databází, screenshotů nebo videozáznamů (pokud je defekt nalezen během provádění testu),
- očekávané a skutečné výsledky,
- rozsah nebo míra dopadu (závažnosti) defektu na zájmy zúčastněných(é) stran(y),
- naléhavost/priorita opravy,
- stav defektu (např. otevřený, odložený, duplicitní, čekající na opravu, čekající na konfirmační testování, znovu otevřený, uzavřený),
- závěry, doporučení a záznamy o schvalování,
- globální problémy, například jiné oblasti, které mohou být ovlivněny změnou vyplývající z defektu,
- historie změn jako je posloupnost činností provedených členy projektového týmu s cílem izolace a opravy defektu a potvrzení korektnosti opravy,
- reference, včetně odkazu na testovací případ, který odhalil problém.

Některé z těchto detailních informací mohou být automaticky vkládány a/nebo spravovány použitím nástrojů pro řízení defektů, např. automatické přiřazení identifikátoru, přiřazení a aktualizaci stavu reportu o defektu v průběhu svého životního cyklu atd. Defekty zjištěné při statickém testování (zejména při revizích) se většinou dokumentují jiným způsobem, např. v zápise z revizní schůzky.

Příklad obsahu reportu o defektu lze nalézt v normě *ČSN ISO/IEC/IEEE 29119-3* (norma však používá termín „report o incidentech“).

6 Nástroje pro podporu testování	**40 minut**

Klíčová slova

Testování řízené daty, testování řízené klíčovými slovy, automatizace testů, nástroj pro provádění testů, nástroj pro management testování.

Studijní cíle

6.1 Základní informace o testovacích nástrojích

FL-6.1.1 (K2) Klasifikovat testovací nástroje podle jejich účelu a podporovaných testovacích činností.
FL-6.1.2 (K1) Určit výhody a rizika automatizace testů.
FL-6.1.3 (K1) Zapamatovat si specifické okolnosti týkající se nástrojů pro provádění testů a nástrojů pro management testování.

6.2 Efektivní využívání nástrojů

FL-6.2.1 (K1) Identifikovat hlavní zásady pro výběr nástroje.
FL-6.2.2 (K1) Zapamatovat si cíle využívání pilotních projektů při zavádění nástroje.
FL-6.2.3 (K1) Identifikovat faktory úspěchu pro vyhodnocení, implementaci, nasazení a průběžnou podporu testovacích nástrojů v rámci organizace.

6.1 Základní informace o testovacích nástrojích

Testovací nástroje lze použít pro podporu jedné nebo více testovacích činností. Základní kategorie nástrojů zahrnují:

- nástroje používané přímo při testování, například nástroje pro (obvykle automatizované) provádění testů nebo nástroje pro přípravu testovacích dat,
- nástroje, které pomáhají spravovat požadavky, testovací případy, testovací procedury, automatizované testovací skripty, výsledky testů a testovací data, dále nástroje pro reportování a monitoring provádění testů,
- nástroje používané k analýze a vyhodnocování,
- nástroje, které jiným způsobem pomáhají při testování (v tomto smyslu je testovacím nástrojem také tabulkový procesor).

6.1.1 Klasifikace testovacích nástrojů

V závislosti na kontextu mohou testovací nástroje naplňovat jeden či více z následujících cílů:

- Zlepšit účinnost testovacích činností automatizací opakujících se úkolů nebo úkolů, které mohou být při ručním provádění nákladné (např. provádění testů při regresním testování).
- Zlepšit účinnost testování podporou manuálních testovacích činností v celém procesu testování (viz kapitola 1.4).
- Zlepšit kvalitu testovacích činností tím, že umožní testovat konzistentně a lépe reprodukovat defekty.
- Automatizovat činnosti, které nemohou být provedeny manuálně (například testování výkonnosti větším počtem souběžných požadavků).
- Zvýšit spolehlivost testů (např. pomocí automatizace porovnávání rozsáhlých množin dat nebo automatizovanou simulací chování).

Nástroje je možné rozdělit podle řady kritérií jako jsou účel, cena, licenční model (např. komerční nebo open source) a použité technologie. V těchto osnovách jsou nástroje rozděleny podle testovacích činností, které podporují.

Některé nástroje podporují výhradně nebo převážně jednu činnost, jiné jich mohou podporovat více (takové nástroje jsou zařazeny pod tu činnost, se kterou jsou nejvíce spojeny). Nástroje od jednoho dodavatele (zejména ty, které byly navrženy jako vzájemně spolupracující) mohou být poskytovány jako jedna integrovaná sada.

Některé typy testovacích nástrojů mohou ovlivňovat výsledek (intrusive tools) – jejich použití může mít dopad na skutečný výsledek testu. Například skutečné odezvy aplikace se mohou lišit kvůli dodatečným instrukcím, které jsou spouštěny pomocí nástroje pro testování výkonu nebo míra pokrytí kódu může být zkreslená kvůli použití nástroje pro měření pokrytí (např. kódu). Důsledky použití intruzivních nástrojů se nazývají efekt měřící sondy.

Některé nástroje jsou vhodnější spíše pro vývojáře než testery (např. nástroje, které se používají během testování komponent a integračního testování). Takové nástroje jsou v následujících kapitolách označeny symbolem D (jako developer).

Nástroje podporující management testování a management testwaru

Tyto nástroje mohou být použity ve všech testovacích aktivitách v průběhu celého životního cyklu vývoje softwaru. Mezi nástroje podporující management testování a management testwaru patří:

- nástroje pro management testování a nástroje pro management životního cyklu aplikací (ALM – application lifecycle management),
- nástroje pro správu požadavků (např. trasovatelnost do objektů testování),
- nástroje pro management defektů,
- nástroje pro konfigurační management,
- nástroje pro průběžnou integraci (D).

Nástroje podporující statické testování

Nástroje pro statické testování jsou spojeny s činnostmi a výhodami uvedenými v kapitole 3. Mezi tyto nástroje patří:

- nástroje pro statickou analýzu (D).

Nástroje podporující návrh a implementaci testů

Nástroje pro návrh testů podporují tvorbu dobře udržovatelných pracovních produktů (zejména při návrhu a implementaci testů), např. testovacích případů, testovacích procedur a testovacích dat. Mezi tyto nástroje patří:

- nástroje pro testování založené na modelu,
- nástroje pro přípravu testovacích dat.

V některých případech mohou nástroje pro podporu návrhu a implementace testů obsahovat také funkcionality pro provádění testů a jejich protokolování, případně mohou nabízet integrační rozhraní, na něž se nástroje podporující provádění a protokolování testů napojí.

Nástroje podporující provádění a protokolování testů

Existuje mnoho nástrojů pro podporu a zlepšování provádění a protokolování testů. Mezi tyto nástroje patří:

- nástroje pro provádění testů (např. pro provádění regresních testů),
- nástroje pro měření pokrytí, např. pokrytí požadavků nebo pokrytí kódu (D),
- sady testovacího vybavení (test harnesses) (D).

Nástroje podporující měření výkonu a dynamickou analýzu

Další skupina obsahuje nástroje nezbytné pro podporu aktivit zátěžového testování (např. aktivit testování výkonu), neboť tyto činnosti nelze účinně provádět manuálně. Mezi tyto nástroje patří:

- nástroje pro testování výkonnosti,
- nástroje pro dynamickou analýzu (D).

Nástroje podporující specifické testovací činnosti

Kromě nástrojů, které podporují obecný proces testování, existuje mnoho nástrojů podporujících specifické testovací činnosti nefunkcionálního testování.

6.1.2 Výhody a rizika automatizace testů

Prosté pořízení nástroje není zárukou úspěchu. Každý nový nástroj zavedený do organizace bude vyžadovat dodatečné náklady pro dosažení reálných a trvalých přínosů. S použitím nástrojů v testování se pojí možné výhody, ale také rizika. Toto platí zejména o nástrojích pro provádění testů (často označovaných jako nástroje pro automatizaci testů).

Mezi možné výhody spjaté s použitím nástrojů na podporu provádění testů patří:

- snížení opakujících se manuálních činností, čímž dochází k úspoře času (například provádění regresních testů, nastavení nebo úklid testovacího prostředí, opětovné zadávání stejných testovacích dat nebo kontrola proti standardům kódování),
- vyšší konzistence a opakovatelnost (např. testovací data jsou vytvořena jednotným způsobem, testy jsou spouštěny pomocí nástroje v stejném pořadí se stejnou frekvencí a jsou důsledně odvozovány na základě požadavků),
- objektivnější vyhodnocení (např. metriky statických metod, míry pokrytí),
- snazší přístup k informacím o testování (např. statistiky a grafy o postupu v testování, o výskytu defektů nebo o výkonnostních charakteristikách).

Mezi možná rizika patří:

- nerealistická očekávání od daného nástroje (včetně funkcionality a snadnosti použití),
- podcenění času, nákladů a pracnosti spojených s prvotním zavedením nástroje (včetně školení a nákladů na externí specialisty),
- podcenění času a pracnosti potřebných k dosažení významných a trvalých přínosů (včetně potřebných změn v procesu testování a průběžného zlepšování způsobu, jakým je nástroj používán),
- podcenění pracnosti potřebné k údržbě testwaru,
- přílišná důvěra v nástroj (ve smyslu náhrady práce testera při navrhování a provádění testů nebo využívání automatizovaného testování v případech, kdy by bylo lepší testovat manuálně),
- zanedbání správy verzí testwaru,
- podcenění problémů spojených s integrací s jinými důležitými nástroji (např. pro správu požadavků, konfigurační management nebo management defektů) a problémů s interoperabilitou nástrojů, které nebyly vytvořeny jedním dodavatelem,
- dodavatel nástroje může ukončit svou obchodní činnost, může přestat nástroj podporovat nebo jej prodat jinému dodavateli,
- nízká úroveň podpory, aktualizací a oprav chyb dodavatelem,
- pozastavení open source projektu,
- nástroj nemusí podporovat nové platformy nebo technologie,
- nejasné vlastnictví nástroje v organizaci (např. není jasné, kdo má poskytovat školení, kdo zajišťuje aktualizace atd.).

6.1.3 Specifické faktory týkající se nástrojů pro provádění testů a nástrojů pro management testování

Pro hladký a úspěšný výběr a implementaci nástrojů pro provádění testů a nástrojů pro management testování v organizaci je třeba zvážit řadu faktorů.

Nástroje pro provádění testů

Nástroje pro provádění testů kontrolují (automaticky ovládají) testované objekty pomocí automatizovaných testovacích skriptů. Pokud má použití tohoto typu nástroje přinést významný přínos, je mnohdy zapotřebí velké pracnosti.

- Přístup zachytávání testů: Zachycení testů pomocí záznamu akcí manuálního testera u nástrojů typu nahraj/přehraj (capture/playback) vypadá na první pohled atraktivně. Tento přístup však není vhodný při velkém počtu testovacích skriptů. Každý nahraný skript je lineární reprezentací akcí manuálního testera s konkrétními daty, aniž by byly společné akce či data vyčleněny do speciálního skriptu. Takto zaznamenané skripty mohou být nestabilní v případě neočekávaných událostí, které nastanou v testovaném objektu (např. zobrazení neočekávaného dialogového okna), nicméně generované skripty stále vyžadují průběžnou údržbu mimo jiné z důvodu kontinuálních změn uživatelského rozhraní systému.
- Přístup s využitím testování řízeného daty: Tento přístup vyčleňuje z testovacího skriptu vstupní testovací data a očekávané výsledky (obvykle do tabulky). Testovací skript dokáže taková vstupní data načíst, a tak je možné jej opakovaně spouštět pro každou variantu dat.
- Přístup s využitím testování řízeného klíčovými slovy: Při tomto přístupu jsou definována klíčová slova popisující akce v testovaném systému (tzv. akční slova, např. Přihlas uživatele, Zadej jméno). Testovací skript je pak složen z těchto klíčových slov a odpovídajících testovacích dat. Při překladu či spuštění testovacího skriptu testovací nástroj zajistí transformaci klíčových slov a přidružených testovacích dat na kód (často v low-level skriptovacím jazyku), který realizuje příslušnou akci v testovaném systému (např. vyplnění položek Uživatelské jméno, Heslo a stisk tlačítka Přihlásit).

Výše uvedené metody vyžadují obvykle velké znalosti skriptovacích jazyků (platí pro testery, vývojáře nebo specialisty na automatizaci testů). Při použití přístupů testování řízeného daty nebo testování řízeného klíčovými slovy mohou testeři bez znalosti skriptovacího jazyka přispět např. vytvořením testovacích dat nebo klíčových slov, které budou následně implementovány příslušnými skripty. Bez ohledu na použitou skriptovací techniku je potřeba pro každý test porovnat očekávané výsledky se skutečnými, a to buď dynamicky (v průběhu provádění testu) nebo ex-post na základě uložených výsledků.

Více informací a příkladů o testování řízeném daty a o testování řízeném klíčovými slovy lze nalézt v učebních osnovách *ISTQB-CTAL-TAE* a dále ve *Fewster 1999* a *Buwalda 2001*.

Nástroje pro testování založeném na modelu (MBT) dovolují využít pro testy funkční specifikaci připravenou ve formě modelu (např. diagram aktivit). Přípravu modelu provádí obvykle návrhář systému. MBT nástroj interpretuje model a vytváří specifikace testovacích případů, které mohou být uloženy v nástroji pro management testování a/nebo prováděny nástrojem pro provádění testů (viz učební osnovy *ISTQB-CTFL-MBT*).

Nástroje pro management testování

Nástroje pro management testování mají často rozhraní pro komunikaci s jinými nástroji nebo funkcionality pro práci s tabulkami vytvářenými v tabulkových procesorech. Je pro to řada důvodů, např.:

- Nutnost přenášet užitečné informace ve formátu, který odpovídá potřebám organizace.
- Potřeba udržovat konzistentní trasovatelnost testwaru proti požadavkům uchovávaným v nástroji pro správu požadavků.
- Potřeba svázat testware s verzemi testovaných objektů uchovávaných v nástroji pro konfigurační management.

Dostupnost komunikačních rozhraní je nutné zvážit zejména při zavádění integrovaného nástroje, např. komplexního nástroje pro management životního cyklu aplikace (ALM). Takové nástroje obsahují nejen modul pro management testování, ale také další moduly uchovávající entity, s nimiž

pracují (často v jiném software) různé další skupiny v rámci organizace (např. harmonogram projektu a informace o rozpočtu).

6.2 Efektivní využívání nástrojů

6.2.1 Hlavní zásady pro výběr nástrojů

Mezi hlavní aspekty, na něž by měl být brán zřetel při výběru nástroje pro organizaci, patří:

- posouzení vyspělosti dané organizace, jejích silných a slabých stránek,
- identifikace příležitostí, kde by mohlo použití nástroje vést ke zlepšení procesu testování,
- pochopení technologií používaných testovanými objekty, aby byl zvolený nástroj s těmito technologiemi kompatibilní,
- pochopení nástrojů pro sestavování a kontinuální integraci, které se již v organizaci používají tak, aby byla zajištěna vzájemná kompatibilita a integrace,
- vyhodnocení nástroje vzhledem k jasným požadavkům a objektivním kritériím,
- zjištění, zda je nástroj k dispozici zdarma na zkušební dobu (a na jak dlouho),
- vyhodnocení kvality dodavatele (z pohledu poskytování školení, podpory i obchodních aspektů) nebo úrovně podpory u nekomerčních nástrojů (např. u open source licencí),
- identifikace potřeb týkajících se zaškolení a průběžné podpory uživatelů při používání nástroje,
- vyhodnocení potřeb školení v obecných dovednostech v oblasti testování (včetně dovednosti automatizovat testy) pro ty, kteří budou s nástrojem (příp. nástroji) přímo pracovat,
- zohlednění výhod a nevýhod různých licenčních modelů (např. komerčních nebo open source),
- odhad poměru nákladů a přínosů založených na konkrétním obchodním případu (je-li požadován).

Závěrečným krokem výběrového procesu by mělo být vyhodnocení ověření konceptu (proof-of-concept). Ověření konceptu probíhá obvykle v rámci stávající infrastruktury organizace včetně zahrnutí software, který bude testován po případné implementaci nástroje. Díky ověření konceptu lze posoudit, zda bude nástroj pracovat efektivně s testovaným softwarem, případně identifikovat potřebné změny v infrastruktuře organizace.

6.2.2 Pilotní projekty při zavádění nástroje do organizace

Po úspěšném ověření konceptu a finálním výběru nástroje začíná obvykle samotné zavádění nástroje do organizace pilotním projektem. Pilotní projekt má následující cíle:

- získání hlubších znalosti o nástroji, detailní pochopení jeho silných i slabých stránek,
- vyhodnocení vhodnosti nástroje s ohledem na stávající procesy a postupy, včetně určení potřebných změn,
- rozhodnutí o způsobech používání, správy a údržby nástroje, způsobech ukládání a údržby testwaru (např. rozhodnutí o konvenci pojmenovávání souborů a testů, výběr standardů kódování, přístupu pro vytváření knihoven a definování modularity testovacích sad),
- posouzení, zda bude dosaženo očekávaných přínosů za rozumnou cenu,

- pochopení metrik, které má nástroj shromažďovat a reportovat včetně příslušné konfigurace nástroje.

6.2.3 Faktory úspěchu při zavádění a používání nástrojů

Mezi faktory úspěšného zavádění a používání nástrojů v rámci organizace patří:

- postupné zavádění nástroje do celé organizace,
- úprava a zlepšování procesů tak, aby zohledňovaly používání nástroje,
- zajištění zaškolení (např. formou koučování a mentorování) a podpory pro uživatele nástrojů,
- definování pokynů pro používání nástroje (např. vnitřní standardy nebo normy pro automatizaci),
- zajištění způsobu získávání informací o skutečném používání nástroje,
- monitorování využívání nástroje a jeho přínosů,
- poskytování průběžné podpory uživatelům nástroje,
- získávání a vyhodnocování zpětné vazby od uživatelů nástroje.

Při zavádění nástroje do organizace je také důležité zajistit, aby byl nástroj technicky a organizačně integrován do životního cyklu vývoje softwaru. Za provoz a podporu nástroje mohou být zodpovědné samostatné útvary uvnitř organizace a/nebo dodavatelé.

Více informací o nástrojích pro provádění testů lze nalézt v *Graham 2012*.

7 Reference

Normy

ČSN ISO/IEC/IEEE 29119-1 (2013) Softwarové a systémové inženýrství – testování softwaru – část 1: Pojmy a definice

ČSN ISO/IEC/IEEE 29119-2 (2013) Softwarové a systémové inženýrství – testování softwaru – část 2: Proces testování

ČSN ISO/IEC/IEEE 29119-3 (2013) Softwarové a systémové inženýrství – testování softwaru – část 3: Dokumentace testování

ČSN ISO/IEC/IEEE 29119-4 (2015) Softwarové a systémové inženýrství – testování softwaru – část 4: 4 Techniky testování

ČSN ISO/IEC 25010, (2011) Systémové a softwarové inženýrství – Požadavky a hodnocení kvality systémů a softwaru (SQuaRE), Modely kvality systémů a softwaru

ČSN ISO/IEC 20246: (2017) Softwarové a systémové inženýrství – Revize pracovních produktů

2.5 UML, Unified Modeling Language – Referenční příručka, http://www.omg.org/spec/UML/2.5.1/, 2017

Dokumenty ISTQB®

Slovník ISTQB®

Slovník ISTQB® (CZ)

ISTQB Foundation Level Overview 2018

ISTQB-CTFL-MBT Foundation Level Model-Based Tester Extension Syllabus

ISTQB-CTFL-AT Foundation Level Agile Tester Extension Syllabus

ISTQB-CTAL-TA Advanced Level Test Analyst Syllabus

ISTQB-CTAL-TTA Advanced Level Technical Test Analyst Syllabus

ISTQB-CTAL-TM Advanced Level Test Manager Syllabus

ISTQB-CTAL-SEC Advanced Level Security Tester Syllabus

ISTQB-CTAL-TAE Advanced Level Test Automation Engineer Syllabus

ISTQB-CTEL-TM Expert Level Test Management Syllabus

ISTQB-CTEL-ITP Expert Level Improving the Test Process Syllabus

Knihy a články

Beizer, B. (1990*) Software Testing Techniques (2e)*, Van Nostrand Reinhold: Boston MA

Black, R. (2017) *Agile Testing Foundations*, BCS Learning & Development Ltd: Swindon UK

Black, R. (2009) *Managing the Testing Process (3e)*, John Wiley & Sons: New York NY

Buwalda, H. et al. (2001) *Integrated Test Design and Automation*, Addison Wesley: Reading MA

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*, Pearson Education: Boston MA

Fewster, M. a Graham, D. (1999) *Software Test Automation*, Addison Wesley: Harlow UK

Gilb, T. a Graham, D. (1993) *Software Inspection*, Addison Wesley: Reading MA

Graham, D. a Fewster, M. (2012) *Experiences of Test Automation*, Pearson Education: Boston MA

Gregory, J. a Crispin, L. (2015) *Agile Testing*, Pearson Education: Boston MA

Jorgensen, P. (2014) *Software Testing, A Craftsman's Approach (4e)*, CRC Press: Boca Raton FL

Kaner, C., Bach, J. a Pettichord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York NY

Kaner, C., Padmanabhan, S. a Hoffman, D. (2013) *The Domain Testing Workbook*, Context-Driven Press: New York NY

Kramer, A., Legeard, B. (2016) *Model-Based Testing Essentials: Guide to the ISTQB® Certified ModelBased Tester: Foundation Level*, John Wiley & Sons: New York NY

Myers, G. (2011) *The Art of Software Testing*, (3e), John Wiley & Sons: New York NY

Sauer, C. (2000). The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research. *IEEE Transactions on Software Engineering*, *Volume 26, Issue 1, pp 1*

Shull, F., Rus, I., Basili, V. July 2000. How Perspective-Based Reading can Improve Requirement Inspections. *IEEE Computer, svazek 33, číslo 7, str. 73-79*

Van Veenendaal, E. (ed.) (2004) *The Testing Practitioner* (Chapters 8–10), UTN Publishers: The Netherlands

Wiegers, K. (2002) *Peer Reviews in Software*, Pearson Education: Boston MA

Weinberg, G. 2008) *Perfect Software and Other Illusions about Testing*, Dorset House: New York NY

Ostatní zdroje (bez přímých referencí v těchto učebních osnovách)

Black, R., van Veenendaal, E. a Graham, D. (2012) *Foundations of Software Testing: ISTQB® Certification (3e)*, Cengage Learning: London UK

Hetzel, W. (1993) *Complete Guide to Software Testing (2e)*, QED Information Sciences: Wellesley MA

Spillner, A., Linz, T. a Schaefer, H. (2014) *Software Testing Foundations (4e)*, Rocky Nook: San Rafael CA

8 Příloha A – Obecné informace k učebním osnovám

Historie dokumentu

Tento dokument tvoří učební osnovy pro základní úroveň certifikace v oblasti testování (ISTQB® Certified Tester Foundation Level), která je první úrovní mezinárodní kvalifikace schválené ISTQB® (www.istqb.org).

Tento dokument byl připraven během června až srpna 2019 pracovní skupinou složenou z členů jmenovaných organizací ISTQB®. Aktualizace byly přidány po revizních vstupech od členů lokálních výborů, kteří již používali Učební osnovy základní verze 2018.

Předchozí verze tohoto dokumentu byla připravována v letech 2014 až 2018 pracovní skupinou složenou ze jmenovaných členů Mezinárodního výboru pro kvalifikaci testování softwaru (International Software Testing Qualifications Board, ISTQB®). Verze 2018 byla nejprve revidována zástupci členských výborů ISTQB®, a následně zástupci vybranými z mezinárodní komunity pro testování softwaru.

Cíle kvalifikace odpovídající certifikaci základní úrovně

- Získat uznání pro testování jako nezbytnou a profesionálně vedenou součást softwarového inženýrství.
- Poskytnout standardní rámec pro kariérní rozvoj testerů.
- Umožnit profesionálním testerům získat uznání u zaměstnavatelů, zákazníků a partnerů, současně zvýšit jejich postavení.
- Propagovat jednotné a osvědčené testovací postupy v rámci všech disciplín softwarového inženýrství.
- Identifikovat oblasti testování, které jsou relevantní a přínosné pro dané odvětví průmyslu.
- Umožnit dodavatelům softwaru najmout certifikované testery, a zveřejněním politiky náboru získat komerční výhodu.
- Poskytnout příležitost testerům nebo osobám se zájmem o testování k získání mezinárodně uznávané kvalifikace v oblasti testování.

Cíle mezinárodní kvalifikace

- Umožnit porovnání znalostí v oblasti testování v různých zemích.
- Usnadnit testerům jejich uplatnění v zahraničí.
- Umožnit nadnárodním a mezinárodním projektům mít jednotné chápání problematiky testování.
- Celosvětově zvyšovat počet kvalifikovaných testerů.
- Jako iniciativa založená mezinárodně mít větší vliv/hodnotu než lokální řešení v rámci jedné země nebo regionu.
- Rozvíjet jednotný mezinárodní soubor znalostí a vědomostí o testování prostřednictvím učebních osnov a terminologie, a zvyšovat úroveň znalostí o testování u všech zúčastněných.
- Propagovat testování jako profesi v dalších zemích.
- Umožnit testerům získat uznávanou kvalifikaci v jejich rodném jazyce.
- Umožnit sdílení znalostí a zdrojů napříč zeměmi.
- Zajistit mezinárodní kvalifikaci a uznání testerů prostřednictvím zapojení více zemí.

Vstupní požadavky pro tuto kvalifikaci

Vstupním kritériem pro účast na zkoušce ISTQB® základní úrovně v oblasti testování softwaru je, že kandidáti mají zájem o testování softwaru. Nicméně, je silně doporučeno, aby kandidáti rovněž:

- Měli alespoň minimální znalosti z vývoje softwaru nebo testování softwaru, například šest měsíců zkušeností ze systémového nebo uživatelského akceptačního testování nebo vývoje softwaru.
- Absolvovali vzdělávací kurz, který byl akreditován podle ISTQB® standardů jednoho z uznaných členských výborů ISTQB®.

Obecné informace a historie základní úrovně certifikátu v oblasti testování softwaru

Nezávislou certifikaci testerů softwaru zahájil ve Velké Británii Zkušební výbor informačních systémů (Information Systems Examination Board, ISEB) založením Výboru pro testování softwaru (Software Testing Board, www.bcs.org.uk/iseb) v roce 1998. v roce 2002 začala v Německu podporovat společnost ASQF německý kvalifikační program pro testery (German Tester Qualification Scheme, www.asqf.de). Tyto učební osnovy jsou založené na osnovách ISEB a ASQF a zahrnují přeorganizovaný, aktualizovaný a rozšířený obsah. Hlavní důraz je kladen na témata, která nejvíce pomáhají testerům po praktické stránce.

Existující certifikát základní úrovně v testování softwaru (např. z ISEB, ASQF nebo z jednoho z uznaných členských výborů ISTQB®) udělený před vydáním tohoto mezinárodního certifikátu se považuje za jemu rovnocenný. Certifikát základní úrovně nemá omezenou platnost a není nutné jej obnovovat. Datum vydání je uvedeno na certifikátu.

V rámci každé ze zúčastněných zemí jsou lokální záležitosti řízeny národním nebo regionálním výborem pro testování softwaru uznaným ISTQB®. Povinnosti členského výboru jsou definovány na úrovni ISTQB® globálně, ale implementovány jsou v rámci každé země. Mezi povinnosti lokálních výborů patří akreditace poskytovatelů školení a zajištění zkoušek.

9 Příloha B – Studijní cíle a kognitivní úroveň znalostí

Pro účely těchto učebních osnov jsou definovány následující studijní cíle. Každé téma v osnovách bude přezkoušeno dle odpovídajícího studijního cíle.

Úroveň 1: Zapamatovat si (K1)

Kandidát je schopen určit, zapamatovat si a vybavit si daný termín nebo pojem.

Klíčová slova: Identifikovat, zapamatovat si, vzpomenout si, vybavit si, určit, znát.
Příklad:

Dokáže určit definici „selhání“jako:

- „Nedodání služby koncovému uživateli nebo jiné zainteresované straně“ nebo
- „Odlišnost komponenty nebo systému od jeho očekávané dodávky, služby nebo výsledku.“

Úroveň 2: Pochopit (K2)

Kandidát dokáže označit příčiny nebo vysvětlení k výrokům vztahující se k tématu, dokáže shrnout, porovnat, klasifikovat, roztřídit a uvést příklady pro koncept testování.

Klíčová slova: Shrnout, zobecnit, abstrahovat, klasifikovat, porovnat, namapovat, odlišit, ilustrovat, interpretovat, převést, znázornit, odvodit, vyvodit, roztřídit, vytvořit modely.

Příklady:

Dokáže vysvětlit důvod, proč by měly být testovací analýza a návrh provedeny co nejdříve:

- Najít defekty, když je jejich odstranění levnější.
- Najít nejdříve nejvýznamnější defekty.

Dokáže vysvětlit podobnosti a rozdíly mezi integračním a systémovým testováním:

- Podobnosti: testované objekty pro integrační testování a systémové testování tvoří více než jedna komponenta a jak integrační testování, tak i systémové testování může zahrnovat nefunkcionální typy testů.
- Rozdíly: integrační testování se zaměřuje na rozhraní a interakce, zatímco systémové testování se zaměřuje na aspekty celého systému jako je zpracování od začátku až do konce (end-to-end).

Úroveň 3: Použít (K3)

Kandidát dokáže zvolit správný koncept či techniku a použít je v daném kontextu.

Klíčová slova: Zavést, provést, používat, sledovat postup, uplatnit postup.

Příklady:

- Dokáže identifikovat hraniční hodnoty pro platné a neplatné třídy ekvivalence.
- Dokáže zvolit testovací případy z diagramu přechodu stavů s cílem pokrýt všechny přechody.

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: Boston MA

10 Příloha C – Poznámky k vydání

Učební osnovy ISTQB® Foundation 2018 V3.1 je minoritní aktualizace verze 2018, ke které byly publikovány samostatné poznámky k vydání (release notes) sumarizující každou kapitolu. Kromě toho byla vydána verze ve změnovém režimu.

Učební osnovy základní úrovně ISTQB® ve verzi 2018 jsou výraznou aktualizací a přepsáním osnov vydaných v roce 2011. Z tohoto důvodu neexistují žádné detailní poznámky na úrovni jednotlivých kapitol a sekcí a přehled hlavních změn je uveden pouze v tomto dokumentu. Mimo to (v samostatném dokumentu k vydání) poskytuje ISTQB® trasovatelnost mezi studijními cíli učebních osnov základní úrovně ve verzi 2011 a studijními cíli učebních osnov základní úrovně ve verzi 2018, kde je navíc zdůrazněno, které studijní cíle byly přidány, aktualizovány nebo odstraněny.

Na začátku roku 2017 bylo evidováno více než 550 000 osob ve více než 100 zemích, kteří absolvovali zkoušku základní úrovně, celosvětově existuje více než 500 000 certifikovaných testerů. Za předpokladu, že všichni z nich si přečetli učební osnovy základní úrovně pro složení zkoušky, jedná se pravděpodobně o nejčtenější dokument v oblasti testování vůbec.

Tato významná aktualizace byla provedena s ohledem k tomuto dědictví s cílem zvýšit hodnotu poskytovanou ze strany ISTQB® dalším 500 000 lidí v globální komunitě testerů.

V této verzi byly upraveny všechny studijní cíle tak, aby byly atomické a poskytovaly jasnou trasovatelnost od každého studijního cíle ke konkrétní sekci/sekcím obsahu (a zkouškovým otázkám), které se vztahují k tomuto studijnímu cíli. Stejně tak je zajištěna zpětná trasovatelnost od sekcí obsahu (a zkouškových otázek) k přidruženému studijnímu cíli. Kromě toho časy přidělené jednotlivým kapitolám jsou nyní realističtější než ty, které obsahovaly učební osnovy ve verzi 2011, přičemž byly užity osvědčených heuristiky a vzorce použité jinými učebními osnovami ISTQB®. Tyto postupy jsou založeny na analýze studijních cílů, který mají být zahrnuty v každé kapitole.

Přestože se jedná o učební osnovy základní úrovně vyjadřující osvědčené postupy a techniky, které obstály ve zkoušce času, přináší tyto osnovy změny s ohledem na modernizaci prezentace materiálu, zejména pokud jde o metody vývoje softwaru (např. Scrum a přístup průběžného nasazování) a technologie (např. Internet věcí). Zároveň došlo k aktualizaci užitých standardů tak, aby byly více relevantní:

1. Norma ČSN ISO/IEC/IEEE 29119 nahrazuje Standard IEEE 829.
2. Norma ČSN ISO/IEC 25010 nahrazuje normu ČSN ISO 9126.
3. Norma ČSN ISO/IEC 20246 nahrazuje IEEE 1028.

Z důvodu dramatického rozšíření portfolia ISTQB® za posledních deset let byly do osnov přidány v případě potřeby rozšiřující křížové odkazy na související materiál obsažený v jiných osnovách ISTQB®. Stejně tak byla provedena revize za účelem sladění s ostatními osnovami ISTQB® a slovníkem pojmů. Cílem je usnadnit čtení, porozumění, učení a překlad se zaměřením na zvýšení praktické užitečnosti a rovnováhy mezi znalostmi a dovednostmi.

Podrobnou analýzu změn provedených v této verzi naleznete v dokumentu ISTQB® Foundation Level 2018 3.1 Release Notes.